Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Kozjek

Didaktični pripomoček za učenje robotike

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn

Ljubljana, 2015

Izvorna koda je dostopna na naslovu https://github.com/kk2882/PPMServoC.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Študent naj v okviru diplomske naloge razvije aplikacijo za mobilni operacijski sistem Android, s katero bo mogoče generirati kodiran signal za dekodirno vezje. Kodiran signal naj se po avdiokanalu prenaša v vezje za dekodiranje signala, ki kodiran signal dekodira in na vsak izhodni priključek vezja pošlje signal, ki mu pripada, glede na vrstni red. V kodiranem signalu bodo signali za krmiljenje 8 servomotorjev hkrati. Kot primer aplikacije naj študent razvije še modul, ki omogoča krmiljenje robotske roke z dvema sklepoma in pisalom. Aplikacija naj bo razvita tako, da omogoča konfiguriranje aplikacije za delovanje z različno strojno opremo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Kozjek, z vpisno številko 63120134, sem avtor diplomskega dela z naslovom:

Didaktični pripomoček za učenje robotike

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luke Šajna,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 11. septembra 2015 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Luki Šajnu, ker me je povezal z as. dr. Alešem Jakličem. Posebej se zahvaljujem as. dr. Alešu Jakliču za idejo, pomoč, objavo članka ter nasvete in usmerjanje pri izdelavi diplomskega dela. Na koncu hvala tudi vsem, ki so kakorkoli pripomogli k uspešno izdelanemu diplomskemu delu, in vsem, ki so me med študijem podpirali in verjeli vame.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADB	Android debug bridge	vmesnik za razhroščevanje
API	application programming interface	vmesnik za programiranje aplikacij
DOF	degrees of freedom	prostostne stopnje
FK	forward kinematics	direktna kinematika
IC	integrated circuit	integrirano vezje
IDE	integrated development environment	integrirano razvojno okolje
IK	inverse kinematics	inverzna kinematika
JDK	Java development kit	komplet za razvoj v Javi
JVM	Java virtual machine	javanski navidezni stroj
JSON	JavaScript object notation	objektna notacija JavaScript
PPM	pulse-position modulation	pulzno-pozicijska modulacija
PWM	pulse-width modulation	pulzno-širinska modulacija
SDK	software development kit	komplet za razvoj programske opreme
RIFF	resource interchange file format	datotečni format za izmenjavo virov
XML	extensible markup language	razširljivi označevalni jezik

Povzetek

V diplomski nalogi sta predstavljena razvoj in implementacija aplikacije za mobilni operacijski sistem Android, razvita v programskem jeziku Java. Aplikacija generira kodiran signal za dekodirno vezje, ki lahko krmili do 9 servomotorjev hkrati, na enem izhodnem avdiokanalu. Poleg funkcionalnosti generiranja kodiranega signala je razvit tudi modul, ki lahko krmili robotsko roko z dvema sklepoma in dvigajočim se pisalom. Razvita aplikacija omogoča spreminjanje nastavitev večine parametrov in se tako lahko prilagodi različni strojni opremi in situacijam. V delu sta poleg razvoja in implementacije aplikacije opisana tudi zgradba in delovanje dekodirnega vezja. Predstavljene so metode in uporabljena teorija, tehnologije in orodja.

Ključne besede: aplikacija Android, pulzno-pozicijska modulacija, didaktični pripomoček, Alpha-shapes, inverzna kinematika, RC-servomotor, učenje robotike.

Abstract

The thesis describes a process of development and implementation of the app for mobile operating system Android which was developed in Java programming language. App generates an encoded signal for a decoding circuit which is able to drive up to nine servo motors simultaneously with one output audio channel. Beside generating an encoded signal, we developed a module to control a robotic arm with two joints and a lift able pen. The developed app allows to alter majority of parameters, which makes app more adaptable to work with different hardware and in different situations. The thesis, beside the app development, describes design of the circuit and how it works, used methods and theory behind, used in the app development and used technologies and tools as well.

Keywords: Android application, pulse-position modulation, didactic tool, Alpha-shapes, inverse kinematics, RC servos, educational robotics.

Poglavje 1

Uvod

Izobraževalna robotika je priljubljen način, da se študenti srečajo, pridobijo izkušnje in znanja s področja znanosti, tehnologije, inženirstva in matematičnih disciplin (ang. STEM - *science, technology, engineering and mathematics*). Mobilne naprave so zelo razširjene med uporabniki in dovolj zmogljive, da se lahko primerjajo z osebnimi računalniki. Prav tako ponujajo kar nekaj senzorjev in ostale strojne opreme, ki jo lahko izkoristimo za obogatitev aplikacij. Zato smo kot didaktični pripomoček razvili aplikacijo Android za krmiljenje servomotorjev, ki so priljubljeni aktuatorji na področju robotike.

Za razvoj aplikacije Android smo se odločili zaradi več razlogov. Glavni razlog za razvoj aplikacije za mobilni operacijski sistem Android je, da jo lahko razvijamo na različnih operacijskih sistemih in z različnimi orodji. Eden izmed razlogov je tudi uporaba programskega jezika Java, ki je primarni programski jezik za razvoj aplikacij Android. Z Javo imamo že kar nekaj izkušenj, tudi znanja, saj smo jo v času šolanja velikokrat uporabljali. Poleg tega smo tudi sami lastniki naprave Android, kar nam omogoča, da aplikacijo lahko testiramo in poganjamo na fizični napravi.

Diplomsko delo je razdeljeno na šest poglavij oziroma vsebinskih sklopov. Začne se s poglavjem Uvod in nadaljuje s poglavjem Tehnologije in orodja, v katerem so predstavljeni mobilni operacijski sistem Android ter tehnologije

in orodja, ki smo jih uporabili pri razvoju aplikacije. V tretjem poglavju je predstavljeno dekodirno vezje, ki smo ga razvili, da z njim krmilimo servomotorje prek signala, ki ga generira mobilna naprava. Sledi poglavje Teoretične osnove, ki predstavi algoritem Alpha-shapes ter teorijo direktne in inverzne kinematike. V petem poglavju je opisan razvoj aplikacije Android. Zadnje poglavje zajema sklepne ugotovitve diplomskega dela in odprte zadeve, ki se lahko še naprej razvijajo. Diplomskemu delu je priložena implementacija algoritma Alpha-shapes. Na koncu se nahajajo še literatura, viri, seznami tabel, kode in slik.

Poglavje 2

Tehnologije in orodja

Poglavje predstavlja tehnologije in orodja, uporabljena pri razvoju. Aplikacijo Android smo razvili v programskem jeziku Java in uporabili integrirano razvojno okolje Android Studio, ki je uradno orodje za razvoj aplikacij Android. Pri razvoju vezja za krmiljenje servomotorjev smo za namen testiranja razvili konzolno aplikacijo, ki generira .wav datoteke. Generirana .wav datoteka vsebuje PPM-signal, ki se generira v odvisnosti od vhodnih argumentov konzolne aplikacije. Konzolno aplikacijo smo razvili v programskem jeziku C++. Seznam uporabljenih tehnologij in orodij:

- programski jezik Java,
- programski jezik C++,
- IDE Android Studio,
- Sonic Visualiser,
- WAV,
- JSON,
- operacijski sistem Android,
- CMake in
- Git.

2.1 Programski jezik Java

Java je splošnonamenski programski jezik za objektno orientirano programiranje [21]. Programski jezik Java je razvil James Gosling. Leta 1995 je bil izdan kot glavna komponenta takratne platforme Sun Microsystems Java. Od januarja 2010 je podjetje Sun Microsystems podružnica podjetja Oracle [21].

Java je bila razvita z namenom, da ima čim manj sistemskih odvisnosti, kar omogoča, da se programi razvijajo le enkrat, za različne operacijske sisteme in arhitekture, brez potrebe po vnovičnem prevajanju programa ali specifičnih spremembah za dotični sistem. Sintaksa Jave je izpeljana iz programskih jezikov, kot sta C in C++, vendar je enostavnejša in z malo nizkonivojskimi funkcijami [21].

Da se javanski programi lahko izvajajo na različnih sistemih, skrbi javanski navidezni stroj - JVM (ang. *Java Virtual Machine*), ki izvaja binarno obliko kode. Za prevajanje izvorne kode v binarno obliko skrbi javanski prevajalnik, ki izvorne kode ne prevaja v strojno kodo ali zbirnik (kot na primer prevajalnika za C ali C++), ampak v binarno obliko.

2.2 Programski jezik C++

C++ je splošnonamenski programski jezik, ki ga je zasnoval Bjarne Stroustrup leta 1983. C++ omogoča programiranje z različnimi programerskimi pristopi: proceduralno, objektno, generično in funkcijsko [11].

Uporablja se predvsem pri sistemskem programiranju, vgrajenih sistemih, sistemih z omejenimi viri s poudarkom na zmogljivosti in učinkovitosti.

Programski jezik C++ je standardiziran s strani organizacije ISO (ang. *International Organization for Standardization*); poznamo kar nekaj standardov jezika. Leta 1998 so C++ standardizirali, prva verzija se je imenovala C++98. Pred izidom prvega standarda je za razvoj jezika skrbel Bjarne Stroustrup. Trenutno najnovejša različica standarda jezika je C++14.

2.3 Integrirano razvojno okolje Android Studio

IDE (ang. *Integrated Development Environment*) Android Studio je uradno razvojno okolje za razvoj aplikacij Android in temelji na platformi IntelliJ IDEA [24]. Android Studio je mogoče namestiti na operacijske sisteme Windows, OS X ter Linux. Za delovanje potrebuje JDK (ang. *Java Development Kit*) verzije 7 ali več in Android SDK (ang. *Software Development Kit*). Pri razvoju aplikacije smo uporabili Android Studio, verzijo 1.2, na operacijskem sistemu Windows 7. Uporaba Android Studia olajša delovni tok razvoja aplikacije Android, saj ponuja poln nabor orodij, ki delo poenostavijo. Za avtomatsko izgradnjo projekta in upravljanje z odvisnostmi uporablja Gradle.

2.3.1 Gradle

Gradle je splošnonamenski sistem za izgradnjo projektov ter avtomatizacijo razvoja [20]. Implementira dobre prakse in funkcionalnosti ostalih podobnih sistemov, kot so: Ant, Ivy, Maven in Gant. Pri razvoju aplikacije smo Gradle uporabili za nalaganje in prevajanje odvisnosti, definiranje ciljne in minimalne verzije Android SDK.

2.3.2 Android SDK

Android SDK omogoča razvijalcem, da lahko razvijajo aplikacije za operacijski sistem Android [19]. Glavne funkcionalnosti, ki jih pridobimo z namestitvijo paketa Android SDK [18]:

- **Knjižnice za razvoj** so najbolj pomemben del paketa Android SDK, saj brez njih ne moremo razvijati aplikacij Android.
- **Dokumentacija** omogoča hiter pregled namena in delovanja različnih simbolov znotraj razvojnega okolja Android Studio.

- **Razhroščevalnik adb** (ang. *Android Debug Bridge*) omogoča razhroščevanje programov znotraj priloženega emulatorja in neposredno na napravah Android, ki jih povežemo prek USB (ang. *Universal Serial Bus*) z računalnikom.

- **Emulator** služi kot virtualna naprava Android, v kateri lahko testiramo in razvijamo našo aplikacijo namesto fizične naprave Android. Pri razvoju aplikacije nismo uporabljali emulatorja, saj je fizična naprava bolj odzivna. Tako smo dosegli hitrejši razvoj in testiranje.

- **Statični analizator kode - lint** se privzeto požene vsakič, ko prevajamo program. Lahko ga poženemo tudi ročno. Statični analizator kode skrbi za potencialne hrošče in možne optimizacije z vidika pravilnosti, varnosti, zmogljivosti, dostopnosti ter uporabnosti.

- **Sistem za zbiranje sporočil - logcat** je orodje, ki skrbi za zbiranje dnevniških in razhroščevalnih sporočil. Orodje omogoča, da lahko spremljamo potek delovanja aplikacije in odkrijemo morebitne težave, saj lahko namesto razhroščevalnika, ki bi med sprehajanjem po kodi upočasnil izvajanje programa, uporabimo statični razred *Log*, s katerim lahko pišemo v dnevnik in odkrijemo težavo že v dnevniških sporočilih.

2.4 Sonic Visualiser

Sonic Visualiser je odprtokodna namizna aplikacija s prijaznim uporabniškim vmesnikom, objavljena pod licenco GNU General Public License V2. Namenjena je analizi, vizualizaciji in pregledovanju vsebine avdiodatotek [5]. V diplomski nalogi smo aplikacijo uporabili za analiziranje in pregledovanje PPM-signala v generiranih .wav datotekah.

2.5 WAV

WAV (ang. *Waveform Audio File Format*) je standardni format podjetij Microsoft in IBM za zapisovanje avdiovsebine [22]. Format za osnovo uporablja generični datotečni format za izmenjavo virov - RIFF (ang. *Resource Interchange File Format*), kar pomeni, da se podatki shranjujejo v označenih kosih [12]. Običajno so .wav datoteke nekompresirane in uporabljajo LPCM-kodiranje (ang. *Linear pulse-code modulation*).

2.6 JSON

JSON (ang. *JavaScript Object Notation*) je standardni format za izmenjavo podatkov v berljivi obliki [23]. Običajno se ga uporablja za izmenjavo podatkov med strežniki in spletnimi aplikacijami, kot alternativo za XML (ang. *Extensible Markup Language*) [23].

Čeprav JSON izvira iz skriptnega programskega jezika JavaScript, je neodvisen od programskega jezika, v katerem se uporablja. Za delo z dokumenti JSON obstaja velik nabor knjižnic, ki so na voljo v različnih programskih jezikih.

Format JSON je opisan v dveh standardih: RFC 7159 in ECMA-404. ECMA opisuje le slovnico, medtem ko RFC 7159 opisuje še vidike varnosti in semantike [23].

2.7 Operacijski sistem Android

Android je mobilni operacijski sistem, ki temelji na jedru Linux. Razvili so ga, da bi ustvarili "pametnejši"operacijski sistem ter konkurenco operacijskemu sistemu Symbian in Microsoft Windows Mobile [16].

Trenutno za njegov razvoj skrbi ameriško multinacionalno tehnološko podjetje Google, ki ga je leta 2005 prevzelo od podjetja Android Inc. ter ustanovilo poslovno združenje več podjetij z imenom OHA (ang. *Open Handset Alliance*) [16]. Združenje so ustanovili z željo po skupnem razvoju odprtih standardov na področju telefonije. Tako želijo približati mobilne naprave uporabnikom z vse cenejšimi in naprednejšimi pametnimi mobilnimi napravami. Glavne prednosti operacijskega sistema Android so:

- enostavnost, odzivnost ter večopravilnost;
- samodejno sinhroniziranje s storitvami Google;
- proizvajalcem mobilnih naprav ni treba razvijati operacijskih sistemov;
- odprotokodnost operacijskega sistema pripomore k cenejšemu in lažjemu razvoju aplikacij;
- večina aplikacij je brezplačnih;
- avtomatsko posodabljanje aplikacij ...

2.7.1 Verzije

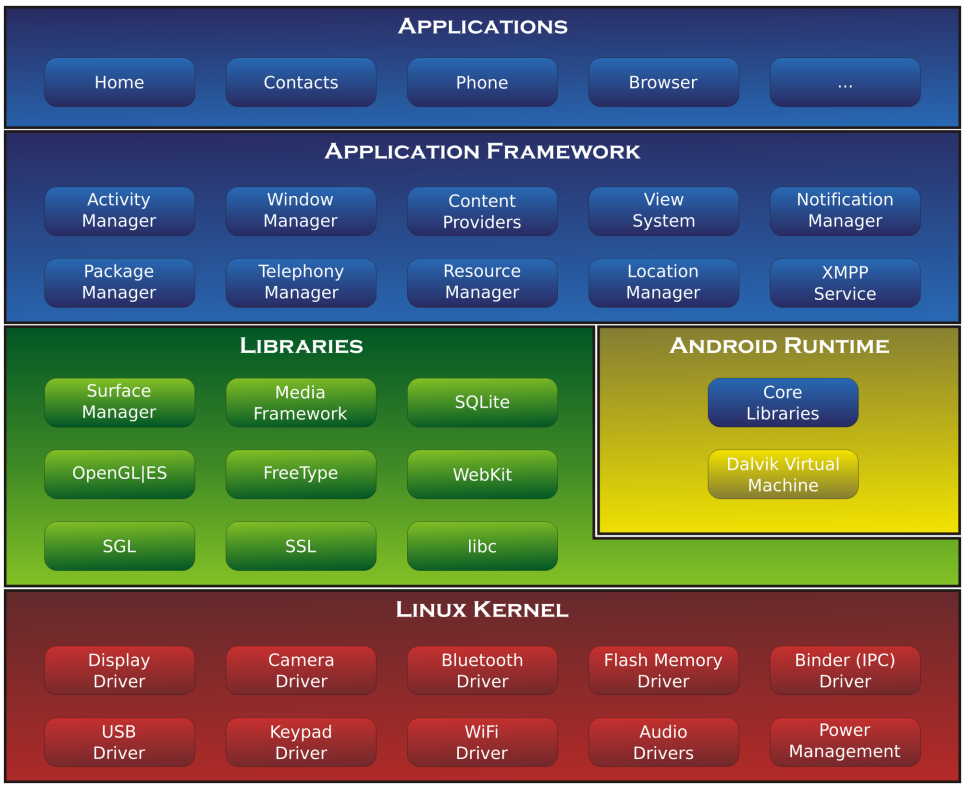
Prva komercialna verzija mobilnega operacijskega sistem Android 1.0 je bila izdana 23. septembra 2008 [15, 16]. Ta verzija je bila zelo okrnjena z malo funkcionalnostmi, vendar je bil to začetek danes zelo popularnega operacijskega sistema na mobilnih in ostalih napravah. Zadnja verzija operacijskega sistema Android je 5.0 Lollipop. Verzije Androida so poimenovali po sladicah, ki si sledijo po abecednem vrstnem redu. Seznam verzij ter njihova imena (Tabela 2.1):

Tabela 2.1: Android verzije

Verzija	Ime	Datum
Android 1.0	/	23. 09. 2008
Android 1.1	/	09. 02. 2009
Android 1.5	Cupcake	27. 04. 2009
Android 1.6	Donut	15. 09. 2009
Android 2.0	Eclair	26. 10. 2009
Android 2.2	Froyo	20. 05. 2010
Android 2.3	Gingerbread	06. 12. 2010
Android 3.0	Honeycomb	22. 02. 2011
Android 4.0	Ice Cream Sandwich	18. 10. 2011
Android 4.1/4.2/4.3	Jelly Bean	09. 07. 2012/13. 11. 2012/ 24. 07. 2013
Android 4.4	Kit Kat	31. 10. 2013
Android 5.0/5.1	Lollipop	12. 11. 2014/09. 03. 2015

2.7.2 Sklad in zgradba operacijskega sistema

Operacijski sistem Android je sestavljen iz petih glavnih delov: aplikacije, aplikacijskega ogrodja, knjižnice, jedra Linux in prevajalnika (Slika 2.1).



Slika 2.1: Sklad operacijskega sistema Android

Jedro Linux

V jedru Linux se nahajajo gonilniki za strojno opremo; skrbi za varnost, delo s pomnilnikom, mrežno komunikacijo in upravlja z energijo.

Prevajalnik

Vsebuje temeljne knjižnice in navidezni stroj Dalvik, ki je optimiziran za poganjanje aplikacij Android na mobilnih napravah [14]. Običajno so aplikacije napisane v programskem jeziku Java in prevedene v binarno obliko, primerno za poganjanje v JVM. Da se lahko prevedena koda izvaja na napravah Android, je treba binarno obliko prevesti še v binarno obliko za navidezni stroj Dalvik. Na novo prevedena koda se shrani v datoteko s končnicama .dex (ang. *Dalvik EXecutable*) in .odex (ang. *Optimized Dalvik EXecutable*).

Knjižnice

Knjižnice omogočajo razvijalcem dostop do komponent strojne opreme, kakršna je na primer grafična procesna enota, do katere lahko dostopamo s knjižnico OpenGL ES (ang. *OpenGL for Embedded Systems*). Vsebuje tudi knjižnico za podatkovno zbirko SQLite, ki je zelo priljubljena pri vgrajenih sistemih.

Aplikacijsko ogrodje

Vsebuje sistemske aplikacije: upravljalca aktivnosti, upravljalca oken, upravljalca obvestil, upravljalca virov, upravljalca lokacije, sistem za poglede ...

Aplikacije

Gre za aplikacije, ki tečeje na operacijskem sistemu Android. Ponavadi so napisane v programskem jeziku Java in XML za definiranje grafičnega vmesnika posameznih aktivnosti ter definiranje manifest dokumenta, v katerem so definirane celotna aplikacija, pravice ter ostale nastavitve. Aplikacije so lahko napisane tudi v programskem jeziku C/C++ ali Go, vendar za te ni tako dobro podprt API (ang. *Application Programming Interface*), da bi jim omogočil enake možnosti, kot če programiramo v programskem jeziku Java [16]. Zato se C/C++ uporabi le takrat, ko je za del programa resnično pomembna hitrost delovanja in enakega učinka ne moremo doseči v programskem jeziku Java.

2.8 CMake

CMake je odprtokodni skupek orodij za prevajanje, testiranje in avtomatizacijo procesa razvoja programske opreme [13]. Deluje na več različnih operacijskih sistemih. Programi omogočajo generiranje avtohtonih skript ali projektov za integrirana razvojna okolja, kot sta na primer Visual Studio in Eclipse. CMake se večinoma uporabljajo na projektih, ki podpirajo več

različnih platform. Tako se poenostavi proces prevajanja, testiranja ter ostalih procesov, saj je treba vzdrževati le eno skripto, ki jo običajno sestavlja več datotek *CMakeFiles.txt*. Pri diplomskem delu smo uporabili CMake 3.1.1 za generiranje *make* skripte za prevajanje in projekta za IDE Visual Studio 2013.

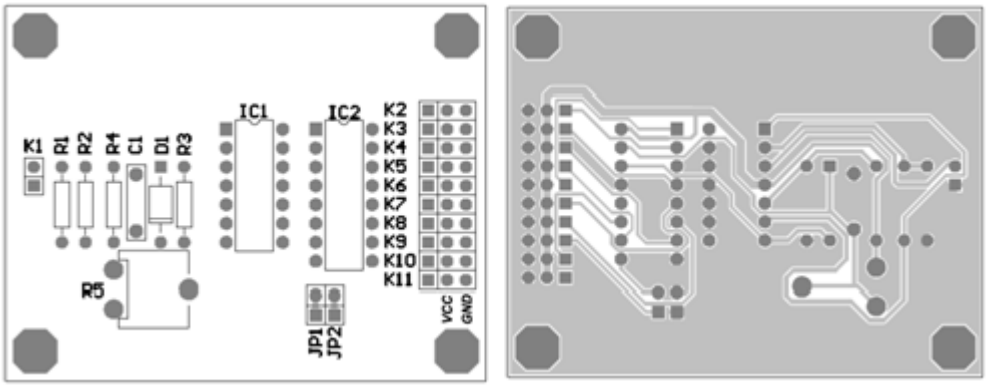
2.9 Git

Git je porazdeljen sistem za podporo obvladovanja verzij (ang. *Distributed Version Control System (DVCS)*), ki ga je leta 2005 razvil Linus Torvalds za potrebe razvoja jedra Linux [17]. Git je postal eden izmed najbolj uporabljenih sistemov za nadzor različic, še posebej za razvoj odprtokodne programske opreme. Git je licenciran pod licenco GNU General Public License verzije 2. Pri razvoju aplikacije smo uporabili Git verzije 1.9.5 za operacijski sistem Windows 7.

Poglavje 3

Vezje

Za krmiljenje servomotorjev smo izdelali tiskano vezje (Slika 3.1), ki omogoča kontroliranje do 9 motorjev hkrati. Običajno se servomotorje krmili programsko, z mikrokontrolerji, ki generirajo PWM-signal (ang. *Pulse Width Modulation*). V diplomskem delu smo se izognili njegovi uporabi in vezje poenostavili. Za krmiljenje več motorjev hkrati prek enega vhodnega signala je treba PWM-signale združiti v skupen signal, ki se imenuje PPM (ang. *Pulse-position modulation*). Za generiranje PPM-signala skrbi aplikacija Android, vezje pa dekodira PPM-signal v PWM-signale za vseh 9 izhodov na vezju.

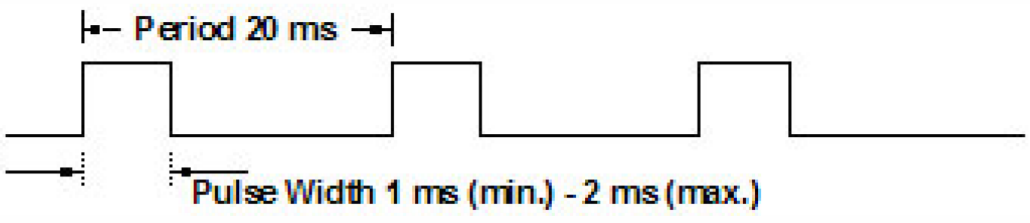


Slika 3.1: Tiskano vezje: levo: zgornja stran plošče tiskanega vezja; desno: spodnja stran plošče tiskanega vezja

3.1 Pulzno-širinska modulacija - PWM

Pulzno-širinska modulacija ali PWM je metoda pulzne modulacije za kodiranje sporočila v pulzni signal [10]. Čeprav se PWM lahko uporablja za kodiranje sporočil pri prenosu po mediju, smo ga v diplomskem delu uporabili za krmiljenje servomotorjev.

Servomotorje se krmili z zaporedjem ponavljajočih se pulzov različnih širin, ki običajno trajajo od 1.0 ms do 2.0 ms (Slika 3.2). Sredinsko lego servomotorja lahko predstavimo s pulzom širine 1.5 ms. Negativno skrajno lego servomotorja predstavlja pulz širine 1.0 ms, pulz širine 2.0 ms pa pozitivno skrajno lego. Nekateri servomotorji omogočajo tudi različne širine pulzov, na primer od 0.7 ms do 2.3 ms, in tako pokrijejo večji kot obračanja.

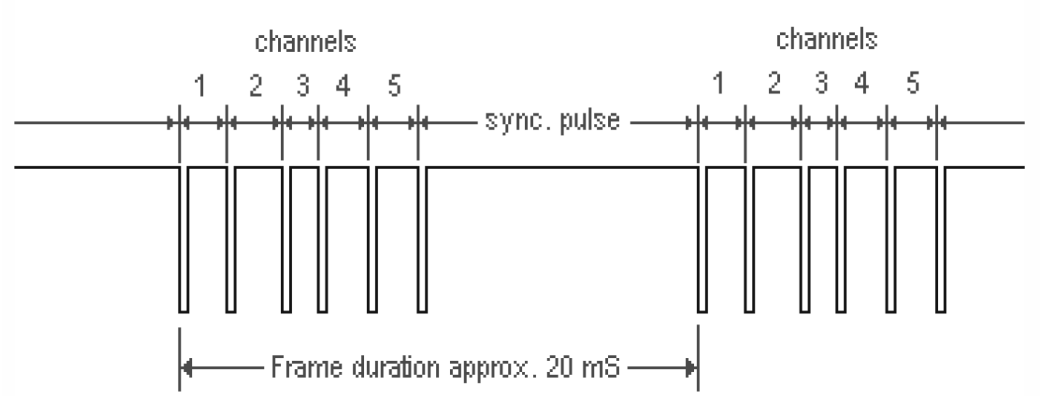


Slika 3.2: PWA-signal za krmiljenje servomotorja

3.2 Pulzno-pozicijska modulacija - PPM

Pulzno-pozicijska modulacija ali PPM je oblika modulacije signala, v kateri se sporočilo dolžine M-bitov kodira v skupen pulz [9].

PPM uporabljamo za kodiranje več PWM-signalov v skupen signal, katerega lahko uporabimo in pošljemo v vezje, kjer se dekodira nazaj v PWM-signale. Na sliki 3.3 je prikazan primer PPM-signala, ki združuje 5 PWM-signalov. PWM-signali so med seboj ločeni z negativnim signalnom dolžine 0.3 ms. Vsak okvir vsebuje združene PWM-signale in sinhronzacijski premor, ki skrbi, da se sprejemnik lahko ponastavi in pripravi na naslednji okvir.



Slika 3.3: PWM-signali, združeni v PPM-signal

3.3 Strojni dekodirnik PPM-signala

Za dekodiranje PPM-signala smo uporabili klasični pristop, s katerim dekodiramo do 9 kanalov. Temelji na čipu CD4017, 5-stopenjskem Johnsonovem števcu [1, 7, 8].

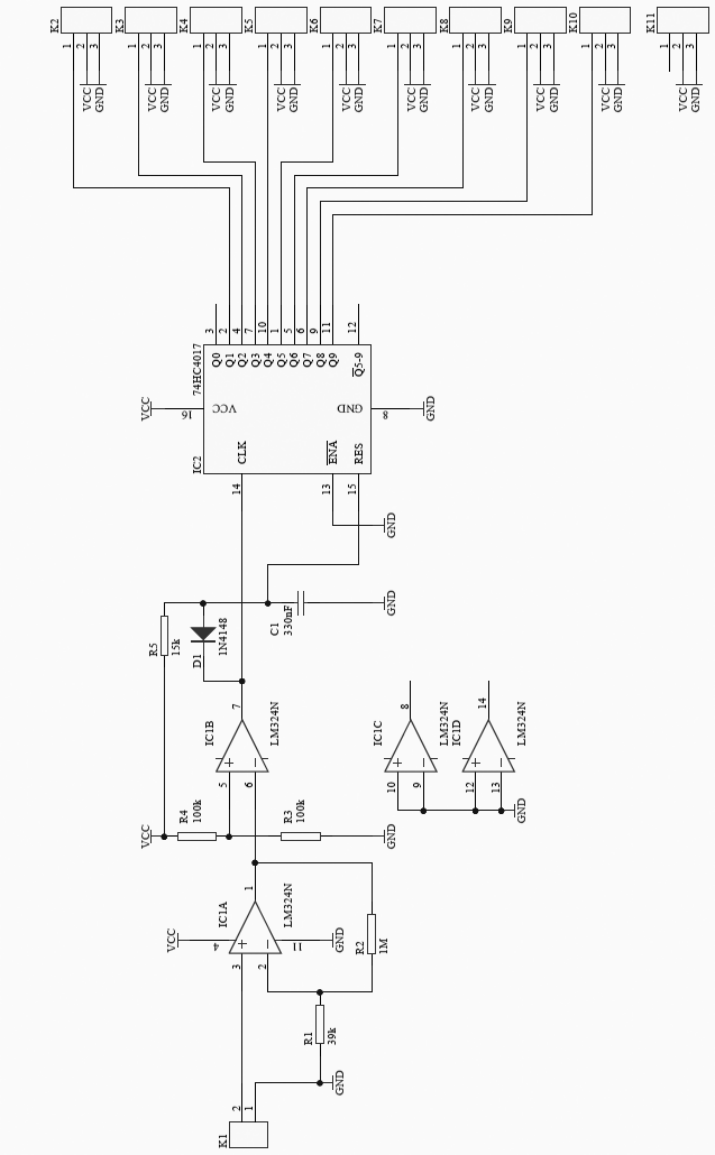
Na sliki 3.4 je prikazana shema vezja, ki smo ga razvili za dekodiranje PPM-signala. Dekodiranje PPM-signala se prične na konektorju K1, skozi katerega v vezje pošiljamo avdiosignal. Potem gre vhodni signal skozi neinvertajoč vhod operacijskega ojačevalnika, ki ga sestavljajo upora R1 in R2 ter operacijski ojačevalnik IC1A.

IC (ang. *Integrated circuit*) ali integrirano vezje je skupek vezij, narejenih iz polprevodnega materiala, kot je silicij. IC je neodvisno vezje v obliki čipa, ki vsebuje različne elektronske komponente, kot so: tranzistorji, operacijski ojačevalniki, logična vrata itd.

Za operacijski ojačevalnik smo uporabili IC LM324N. Ojačan signal iz izhoda 1 na IC1 pošljemo v primerjalnik, ki ga sestavljajo upora R3 in R4 ter operacijski ojačevalnik IC1B. S primerjalnikom invertiramo signal, ki ga uporabimo kot urin signal IC2. Invertiran signal uporabljamo tudi za krmiljenje vezja za ponastavljanje IC2, ki ga sestavljajo upor R4, kondenzator C1 in dioda D1. Izhodni signal IC2 je dekodiran PWM-signal, primeren za krmiljenje servomotorja. Dekodiran signal se razporedi po IC2-izhodih od

Q1 do Q9, ki so povezani na konektorje od K2 do K10. Za napajanje vezja so potrebne 4 AA-baterije, ki se priklopijo na konektor K11.

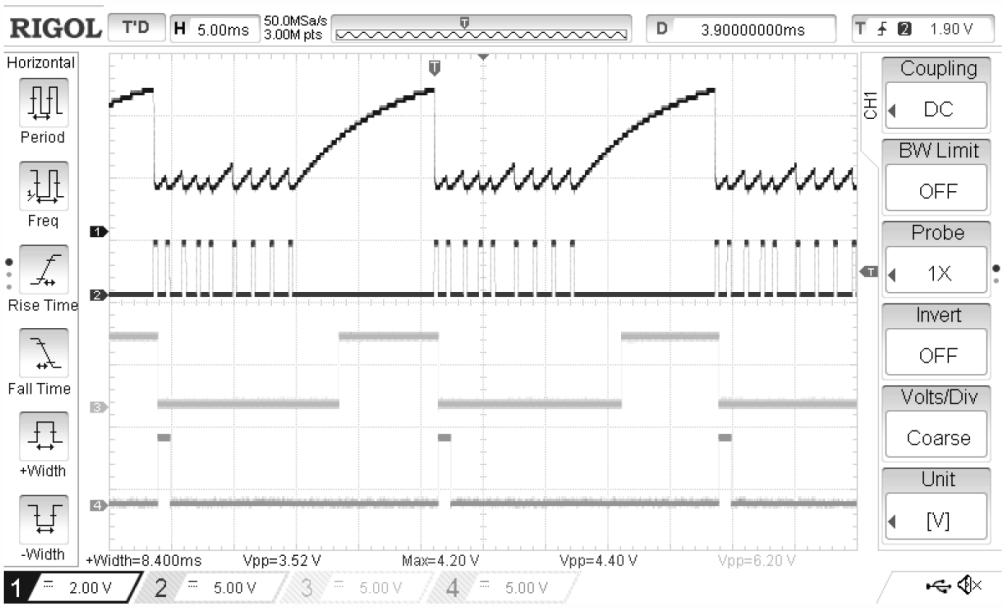
Kot lahko opazimo na sliki 3.4, je izhodov za krmiljenje servomotorjev 9 in ne samo 8, kot je bilo sprva načrtovano. V tej konfiguraciji vezje podpira krmiljenje servomotorjev na vseh 9 izhodih. Prav tako tudi razvita aplikacija podpira generiranje PPM-signala za vseh 9 servomotorjev.



Slika 3.4: Shema vezja

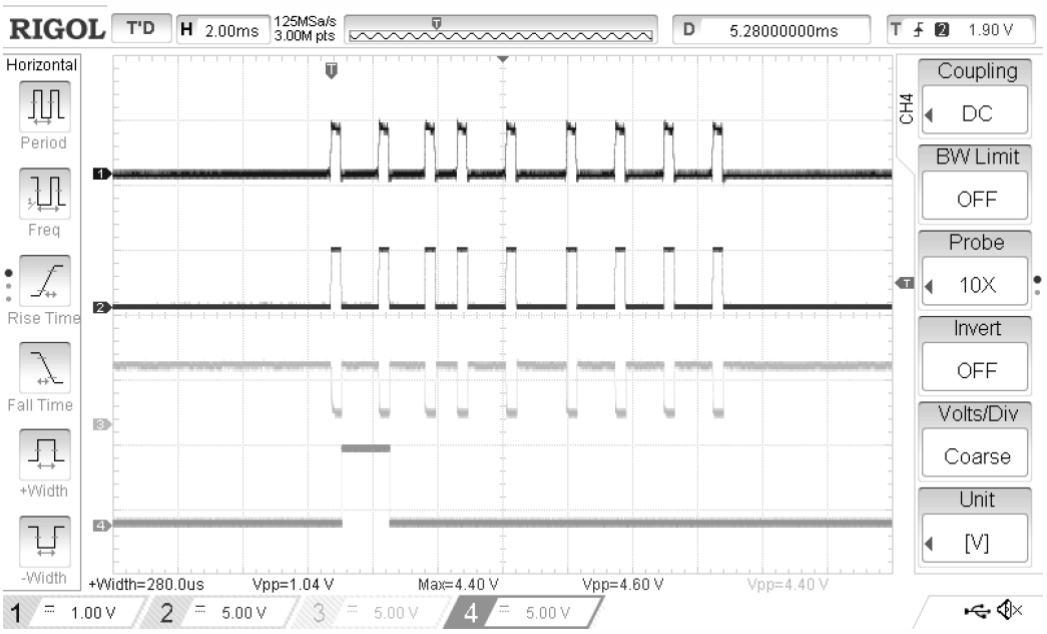
3.4 Signali

Vezje, ki smo ga razvili, ponuja veliko možnosti opazovanja analognih in digitalnih signalov, ki so primerni za opazovanje na osciloskopu. Za primer vzemimo proces polnjenja in praznjenja kondenzatorja C1 v vezju, ki skrbi za ponastavitev IC2, kjer lahko opazujemo, kako se spreminja napetost skozi čas (Slika 3.5).



Slika 3.5: Polnjenje in praznjenje kondenzatorja

Na sliki 3.6 vidimo zaslonsko sliko osciloskopa, ki prikazuje signale v vezju. Na kanalu 1 je signal iz vhoda 3 na IC1A, na kanalu 2 je ojačan signal iz izhoda 1 na IC1A, na kanalu 3 je invertiran signal IC1A.



Slika 3.6: Signali v vezju

Ojačitev A neinvertiranega operacijskega ojačevalnika je definirana z enačbo (3.1).

$$A = 1 + \frac{R_2}{R_1} \tag{3.1}$$
$$A = 1 + \frac{10^6\Omega}{39 * 10^3\Omega}$$
$$A = 26.64$$

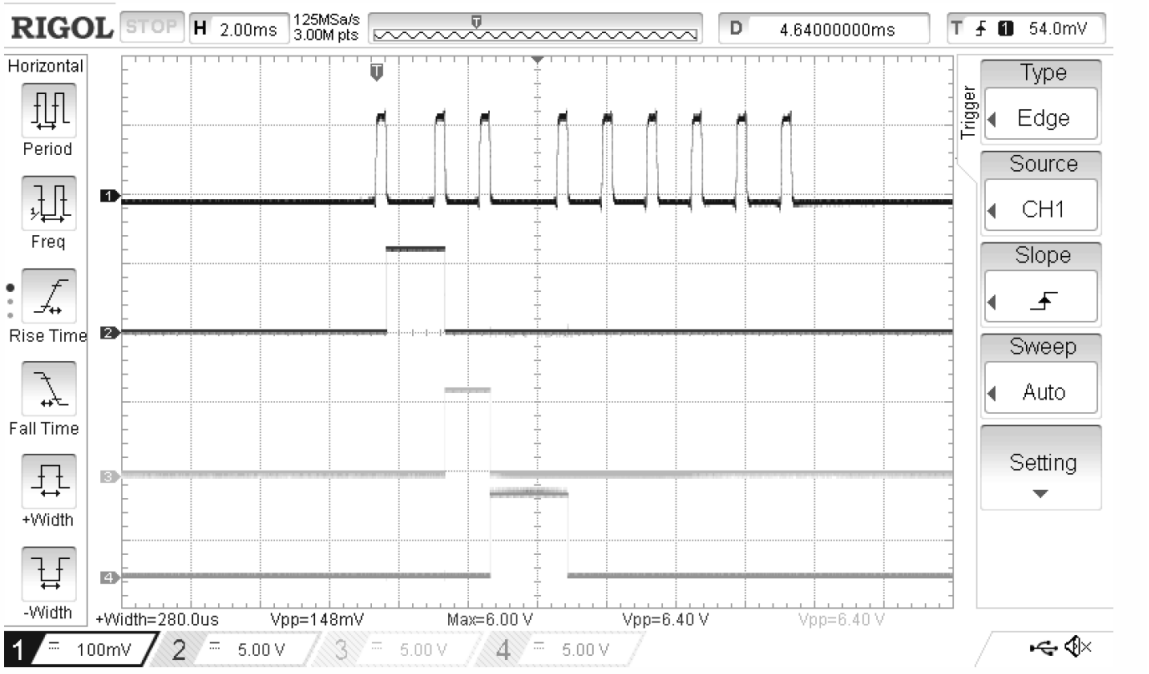
Delilnik napetosti, ki ga sestavljata upora R3 in R4, nastavi vhodno napetost vhoda 5 na IC1B na 3 V, kar je polovica vhodne napetosti VCC, ki je 6 V. Tako primerjalnik lahko učinkovito invertira vhodni signal na vhodu 5 v IC1B. Če je vhodna napetost vhoda 6 na IC1B nižja od 3 V, je izhod primerjalnika skoraj enak vhodni napetosti VCC; če je vhodna napetost nad 3 V, je izhodna napetost 0 V. Invertiran signal lahko vidimo na sliki 3.6, na 3. kanalu.

Za ustrezno dekodiranje posameznega okvirja je pred vsakim novim prihajajočim okvirjem treba ponastaviti števec CD4017. Ponastavljanje dosežemo s sinhronizacijskim premorom, ki se nahaja na koncu vsakega okvirja. Napetost na RES-vhodu v IC2 ima enako napetost kot kondenzator C1, ki se polni skozi upornik R5, z maksimalno časovno konstanto, ki je definirana z enačbo (3.2).

$$\begin{aligned} \tau &= RC \\ \tau &= R5 * C1 \\ \tau &= 47 * 10^3\Omega * 330 * 10^-9F \\ \tau &= 47 * 10^3\Omega * 330 * 10^-9\frac{s}{\Omega} \\ \tau &= 15.51ms \end{aligned} \tag{3.2}$$

Na koncu vsakega okvirja se nahaja premor, ki zaradi odsotnosti negativnega pulza omogoči, da se kondenzator C1 začne polniti. Ob zadostni napetosti se signal na vhodu RES na IC2 sproži in ponastavi števec ter pripravi za naslednji okvir. Polnjenje in praznenje kondenzatorja C1 lahko vidimo na sliki 3.5. Na 1. kanalu je prikazan signal na vhodu RES, na 2. kanalu je prikazan izhodni signal, na izhodu 1 iz operacijskega ojačevalnika IC1A, kjer lahko vidimo ojačan vhodni signal iz mobilne naprave. Na 3. in 4. kanalu sta prikazana izhoda iz IC2, Q1 in Q2.

Ker prag napetosti, ki je potreben za proženje ponastavitve na IC2, ni vedno enak in se razlikuje med različnimi IC, smo za upor R5 uporabili potenciometer. Tako lahko z osciloskopom in potencimetrom natačno nastavimo čas polnjenja, ki je potreben za doseg praga, ki ponastavi števec. Na sliki 3.7 vidimo, kako se vhodni PPM-signal na 1. kanalu dekodira in razdeli na prve tri izhode Q1, Q2 in Q3, ki so povezani s servomotorji.



Slika 3.7: Vhodni PPM-signal in dekodirani PWM-signali

Poglavje 4

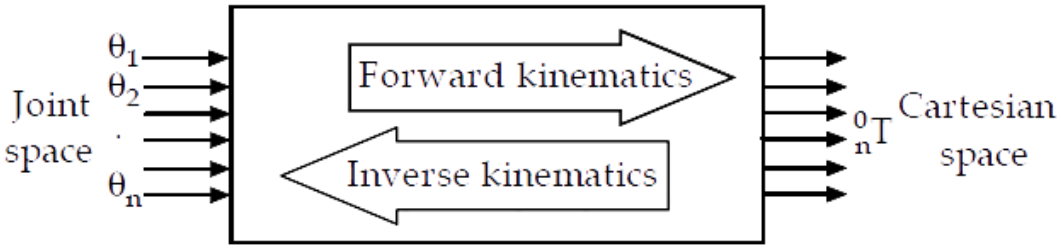
Teoretične osnove

V tem poglavju bodo opisane teoretične osnove, ki smo jih uporabili pri izdelavi diplomskega dela.

4.1 Kinematika

Kinematika je veda, ki preučuje gibanje teles, ne da bi se spraševala po vzroku, kot je na primer delovanje sil [3]. Kinematiko robota lahko delimo na direktno kinematiko ali FK (ang. *Forward Kinematics*) in inverzno kinematiko ali IK (ang. *Inverse Kinematics*). FK in IK sta med seboj povezana, kot prikazuje slika 4.1. FW preslikuje kote sklepov v kartezični koordinatni sistem. IK preslikuje koordinate kartezičnega koordinatnega sistema v kote sklepov. FK je nekompleksen in enostaven problem, saj rešitev FK vedno obstaja. Izračun IK pa je kompleksnejši problem, saj je računsko zahtevnejši in časovno obsežnejši.

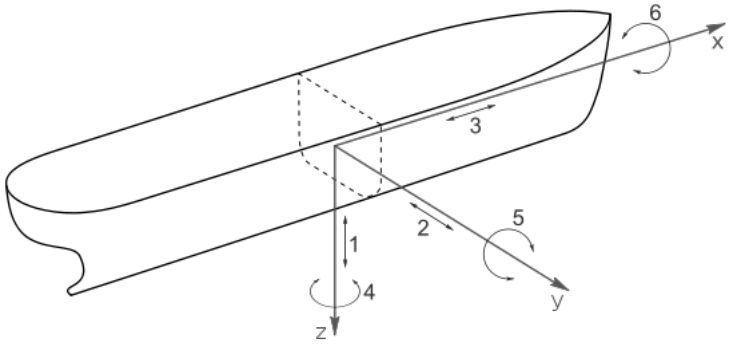
V diplomskem delu smo uporabili IK za izračun lege konice robotske roke, ki jo sestavljajo trije sklepi. FK smo uporabili za preverjanje rešitev inverzne kinematike ter za določitev delovnega prostora za namen aplikacije. Implementacija kinematike je predstavljena v poglavju 5.3.4, ki opisuje razviti modul, kjer se z mobilno napravo krmili robotsko roko, ki uporablja dva sklepa za premikanje roke in še en sklep za dvigovanje peresa. Ker je mo-



Slika 4.1: Povezava med direktno in inverzno kinematiko

dul aplikacije razvit za planarni 2-DOF (ang. *Degree of freedom*) robotski manipulator, so enačbe v nadaljevanju izpeljane za 2 sklepa.

DOF ali prostostna stopnja v robotiki pomeni število neodvisnih parametrov, ki so potrebni za opis robotskega manipulatorja v prostoru [6]. Togo telo v prostoru se opiše z lego, ki je predstavljena s 6 prostostnimi stopnjami (Slika 4.2). Prve 3 prostostne stopnje opisujejo pozicijo telesa, preostale 3 pa rotacijo. Robotski sklepi so rotacijski ali translacijski in imajo imajo eno samo prostostno stopnjo [4].



Slika 4.2: 6 prostostnih stopenj na primeru ladje

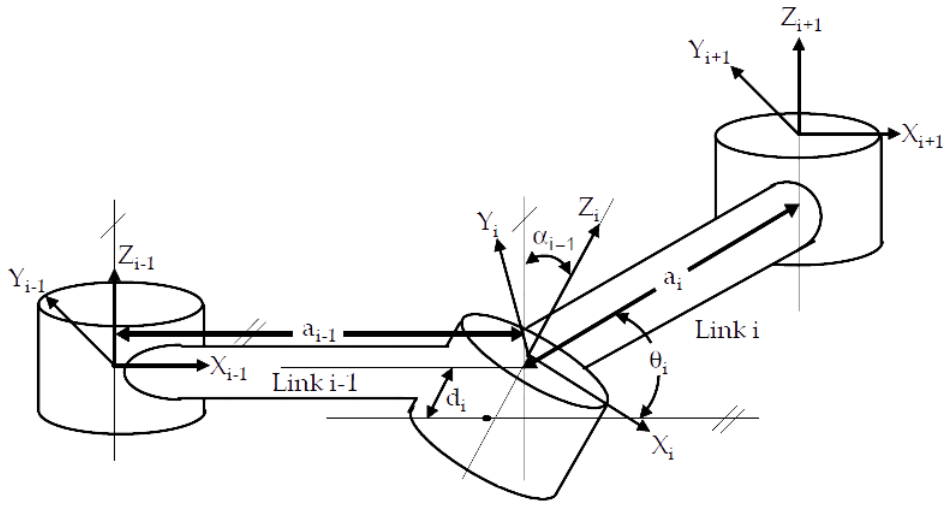
Rotacijski sklep omejuje gibanje dveh sosednjih segmentov na rotacijo. Relativni položaj med segmentoma merimo kot kot zasuka okrog osi sklepa [4, 3].

Translacijski sklep omejuje gibanje dveh sosednjih segmentov na translacijo. Relativni položaj med segmentoma merimo kot razdaljo vzdolž osi sklepa [4, 3].

4.1.1 Direktna kinematika

Cilj izračuna direktne kinematike je končna lega (pozicija in orientacija) robotskega manipulatorja glede na podano konfiguracijo. FK lahko izračunamo na več načinov.

Generični in najbolj uporabljen pristop za reševanje problema FK uporablja metodo Denavit-Hartenberg, ki za opisovanje posameznega sklepa potrebuje štiri parametre. Običajno so označeni: a_{i-1}, α_{i-1}, d_i in θ_i (Slika 4.3).



Slika 4.3: Spremenljivke za splošni manipulator FK

Opis parametrov:

- a_{i-1} predstavlja razdaljo med Z_{i-1} do Z_i glede na os X_{i-1},
- α_{i-1} predstavlja zasuk kota med Z_{i-1} do Z_i glede na os X_i,
- d_i predstavlja razdaljo med X_{i-1} do X_i glede na os Z_i in
- θ_i predstavlja zasuk kota med X_{i-1} do X_i glede na os Z_i.

Za izračun končne lege manipulatorja je treba izračunati homogeno transformacijsko matriko A, ki je sestavljena iz 4 osnovnih transformacij. Izračun matrike za posamezno povezavo med sklepoma:

$$A_i = T_{Z,d_i} R_{Z,\theta_i} T_{X,a_{i-1}} R_{X,\alpha_{i-1}} \tag{4.1}$$

$$\begin{aligned} A_i = &\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & 0 \\ 0 & sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.2}$$

$$A_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_{i-1}) & sin(\theta_i)sin(\alpha_{i-1}) & a_{i-1}cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_{i-1}) & -cos(\theta_i)sin(\alpha_{i-1}) & a_{i-1}sin(\theta_i) \\ 0 & sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Za primer 2-DOF-manipulatorja, za katerega smo implementirali aplikacijo, bomo izračunali transformacijsko matriko A.

Najprej je treba napisati parametre za vse sklepe, da jih bomo pozneje lahko samo vnesli v matriko (4.3). Vrednosti parametrov bomo zapisali v tabelo (Tabela 4.1).

Tabela 4.1: Parametri za izračun FK z metodo Denavit-Hartenberg

Povezava	θ_i	α_{i-1}	a_{i-1}	d_i
1	60°	0°	17 cm	0 cm
2	30°	0°	13 cm	0 cm

Ko so parametri pripravljeni, lahko izračunamo transformacijsko matriko za vsako posamezno povezavo. To naredimo tako, da podatke iz tabele (Tabela 4.1) vstavimo v enačbo matrike A_i (4.3).

$$A_1 = \begin{bmatrix} cos(60) & -sin(60) & sin(60) & 17 * cos(60) \\ sin(60) & cos(60) & -cos(60) & 17 * sin(60) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$A_2 = \begin{bmatrix} cos(30) & -sin(30) & sin(30) & 13 * cos(30) \\ sin(30) & cos(30) & -cos(30) & 13 * sin(30) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

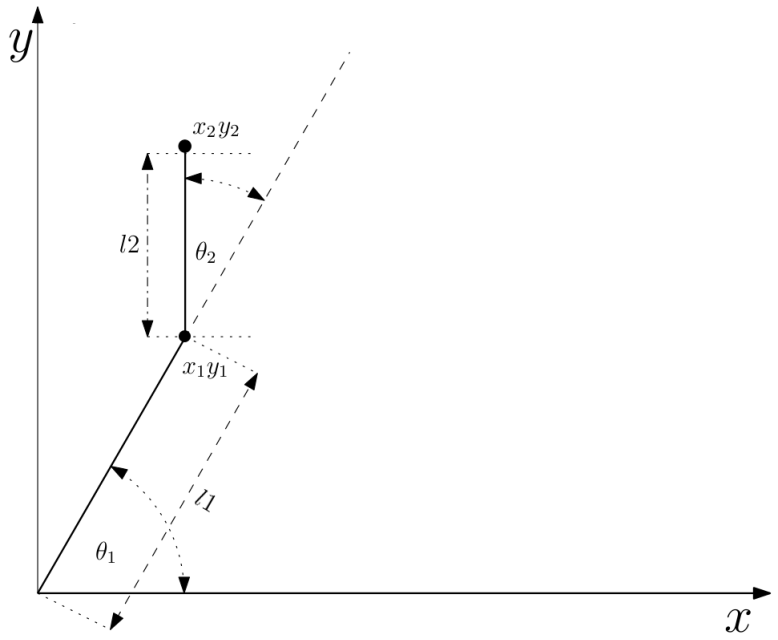
Ko imamo pripravljeni matriki A_1 in A_2, ju zmnožimo in dobimo transformacijsko matriko A, ki jo sestavljata rotacijska matrika R in translacijska matrika T. Za rešitev FK-planarnega 2-DOF-manipulatorja nas zanima matrika T, natančneje komponenti T_x in T_y.

$$A = A_1 A_2 = \left[\begin{array}{ccc|c} & R & & \begin{matrix} T_x \\ T_y \\ T_z \end{matrix} \\ \hline 0.00 & 0.00 & 0.00 & 1.00 \end{array}\right] \tag{4.6}$$

$$A = \left[\begin{array}{ccc|c} 0.00 & -1.00 & 0.00 & 8.50 \\ 1.00 & 0.00 & 0.00 & 27.72 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ \hline 0.00 & 0.00 & 0.00 & 1.00 \end{array}\right] \tag{4.7}$$

$$T = \begin{bmatrix} 8.50 \\ 27.72 \\ 0.00 \\ 1.00 \end{bmatrix} \tag{4.8}$$

Tako dobimo rešitev, da se vrh robotskega manipulatorja nahaja na koordinatah $x = 8.50\ cm$, $y = 27.72\ cm$.



Slika 4.4: Spremenljivke za planarni 2-DOF-manipulator FK

Če računamo FK za preprost manipulator, kot je planarni 2-DOF-manipulator, lahko uporabimo tudi geometrijski pristop. Na sliki 4.4 so označene spremenljivke, ki jih potrebujemo za izračun FK z geometrijskim pristopom. Končno lego planarnega robotskega manipulatorja lahko definiramo z enačbo (4.9) za os x in (4.10) za os y. Parametra, ki ju potrebujemo za izračun FK, sta relativni kot zasuka sklepa, ki je označen z θ_j, in dolžina povezave med sklepoma, ki je označena z l_i.

$$x_n = \sum_{i=1}^{n}(l_i cos(\sum_{j=1}^{i}\theta_j)) \tag{4.9}$$

$$y_n = \sum_{i=1}^{n}(l_i sin(\sum_{j=1}^{i}\theta_j)) \tag{4.10}$$

Tabela 4.2: Parametri za izračun FK z geometrijskim pristopom

Povezava	θ_i	l_i
1	60°	17 cm
2	30°	13 cm

Za primer izračuna FK z geometrijskim pristopom bomo uporabili iste podatke kot pri prejšnjem primeru in jih zapisali v tabelo (Tabela 4.2).

Sedaj lahko podatke iz tabele 4.2 vstavimo v enačbo (4.9) za koordinato na osi x in (4.10) za koordinato na osi y. Izračunali bomo pozicijo točke na koncu povezave z indeksom 2.

$$x_2 = \sum_{i=1}^{2}(l_i cos(\sum_{j=1}^{i}\theta_j)) \tag{4.11}$$

$$x_2 = l_1 * cos(\theta_1) + l_2 * cos(\theta_1 + \theta_2) \tag{4.12}$$

$$x_2 = 17 * cos(60) + 13 * cos(60 + 30) \tag{4.13}$$

$$x_2 = 8.50\ cm \tag{4.14}$$

$$y_2 = \sum_{i=1}^{2}(l_i sin(\sum_{j=1}^{i}\theta_j)) \tag{4.15}$$

$$y_2 = l_1 * sin(\theta_1) + l_2 * sin(\theta_1 + \theta_2) \tag{4.16}$$

$$y_2 = 17 * sin(60) + 13 * sin(60 + 30) \tag{4.17}$$

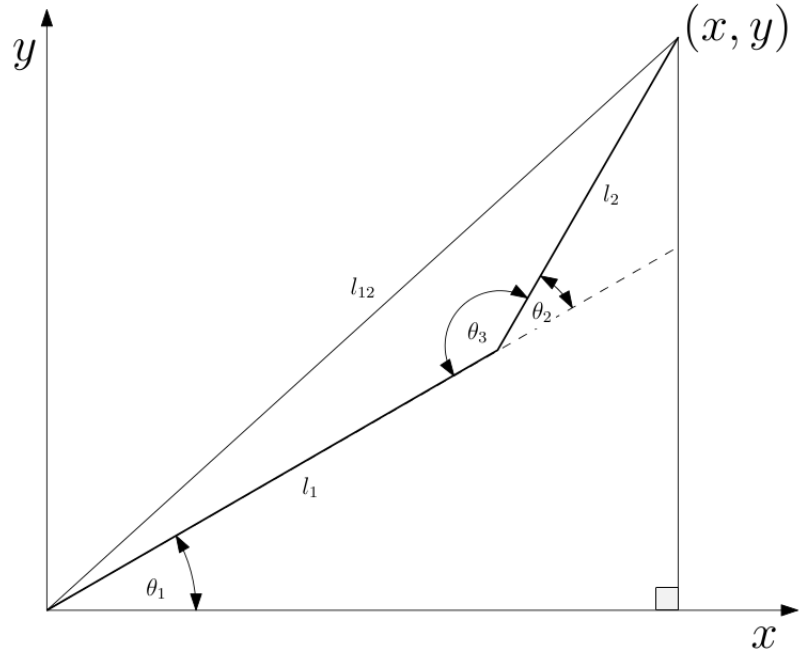
$$y_2 = 27.72\ cm \tag{4.18}$$

Kot vidimo iz (4.14) in (4.18), smo dobili enako rešitev kot v primeru, ko smo FK računali z metodo Denavit-Hartenberg.

4.1.2 Inverzna kinematika

IK deluje ravno obratno kot FK. Cilj IK je pretvorba iz kartezičnega koordinatnega sistema v prostor sklepov, ki je predstavljen s koti. Vhodni podatek za izračun IK je lega, v kateri želimo, da se nahaja vrh robotskega manipulatorja. Pri izračunu IK lahko dobimo eno, več ali 0 rešitev. Rešitev ne obstaja takrat, ko je želena lega izven fizičnega dosega robotskega manipulatorja. Več rešitev je možnih, ko v različnih konfiguracijah kotov sklepov lahko dosežemo isto točko. Takrat moramo preučiti, katera rešitev je za nas najboljša. Običajno želimo, da je premik iz prejšne v novo lego čim krajši; tako vzamemo za mero kar razliko razdalj med trenutnimi in predlaganimi novimi koti.

Tako kot FK se tudi IK lahko izračuna na več načinov. Pogosto se uporablja Jacobijevo matriko in Jacobijevo tehniko za izračun IK (ang. *Jacobian inverse technique*) [3]. V diplomski nalogi smo zaradi preprostosti robotskega manipulatorja uporabili geometrijski pristop.



Slika 4.5: Spremenljivke za planarni 2-DOF-manipulator IK za izračun θ_2

Znani podatki so: l_1, l_2, x in y (Slika 4.5). Izračunati moramo kota sklepov: θ_1 in θ_2. Najprej bomo izračunali kot θ_2. Kot lahko vidimo na sliki 4.5, je kot $\theta_2 = \pi - \theta_3$. Za izračun kota, ki ga omejujeta stranici l_1 in l_2, bomo uporabili kosinusni izrek (4.20). Preden lahko podatke vnesemo v enačbo (4.20), moramo izračunati še dolžino stranice l_{12} (4.19).

$$l_{xy}^2 = x^2 + y^2 \tag{4.19}$$

$$l_{xy}^2 = l_1^2 + l_2^2 - 2l_1 l_2 cos(\theta_3) \tag{4.20}$$

Sedaj imamo pripravljene vse podatke, da lahko izračunamo kot θ_3. Enačbi (4.19) in (4.20) bomo združili in izpostavili $cos(\theta_3)$. Vendar za rešitev IK potrebujemo kot θ_2. Zato bomo pri združitvi zgornjih dveh enačb uporabili še lastnost simetričnosti kosinusa (4.22).

$$\theta_3 = \pi - \theta_2 \tag{4.21}$$

$$cos(\theta_3) = cos(\pi - \theta_2) = -cos(\theta_2) \tag{4.22}$$

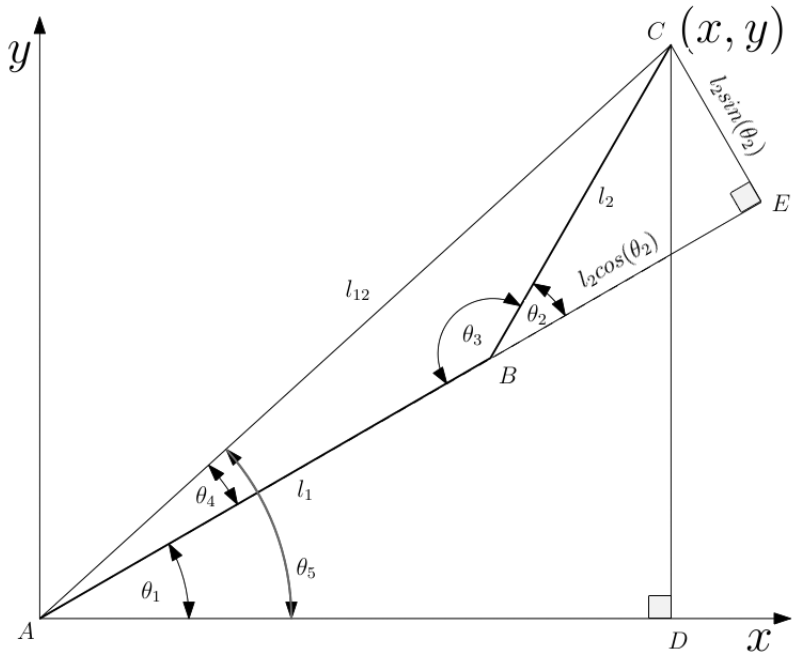
Potem združimo enačbe (4.19), (4.20) in (4.22) v enačbo (4.23). Izpostavimo še $cos(\theta_2)$ (4.24) in izračunamo kot θ_2 (4.25).

$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 cos(\theta_2) \tag{4.23}$$

$$cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2} \tag{4.24}$$

$$\theta_2 = arccos(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}) \tag{4.25}$$

Zdaj lahko izračunamo še kot θ_1, ki bo rezultat razlike kota θ_5 v trikotniku ADC in θ_4 v trikotniku AEC (Slika 4.6). Kot θ_4 bomo izračunali z enačbo (4.26).



Slika 4.6: Spremenljivke za planarni 2-DOF-manipulator IK za izračun θ_1

$$tan(\theta_4) = \frac{l_2 sin(\theta_2)}{l_1 + l_2 cos(\theta_2)} \tag{4.26}$$

Kot θ_5 bomo izračunali z enačbo (4.27).

$$tan(\theta_5) = \frac{x}{y} \tag{4.27}$$

Sedaj, ko imamo podatke za kota θ_4 in θ_5, moramo izračunati še razliko med kotoma (4.29).

$$tan(\theta_1) = tan(\theta_5 - \theta_4) \tag{4.28}$$

$$tan(\theta_1) = \frac{tan(\theta_5) - tan(\theta_4)}{1 + tan(\theta_5)tan(\theta_4)} \tag{4.29}$$

Vstavimo θ_4 iz (4.26) in θ_5 iz (4.27) v (4.29).

$$tan(\theta_1) = \frac{\frac{x}{y} - \frac{l_2 sin(\theta_2)}{l_1 + l_2 cos(\theta_2)}}{1 + \frac{x}{y}\frac{l_2 sin(\theta_2)}{l_1 + l_2 cos(\theta_2)}} \tag{4.30}$$

Ulomek na desni strani, v enačbi (4.30), poenostavimo tako, da imenovalec in števec pomnožimo z $x(l_1 + l_2 cos(\theta_2))$ (4.31).

$$tan(\theta_1) = \frac{y(l_1 + l_2 cos(\theta_2)) - x(l_2 sin(\theta_2))}{x(l_1 + l_2 cos(\theta_2)) + y(l_2 sin(\theta_2))} \tag{4.31}$$

Ko imamo pripravljeni enačbi za izračun IK-planarnega 2-DOF-robotskega manipulatorja, enačbi preverimo še s primerom. Vzamemo iste podatke za dolžino $l_1 = 17\ cm$ in $l_2 = 13\ cm$ kot pri izračunu za FK. Za točko vrha robotskega manipulatorja vzamemo rešitev FK iz prejšnjega poglavja $x = 8.50\ cm$ in $y = 27.72\ cm$. Podatke vstavimo v enačbo (4.25) za kot θ_2 in enačbo (4.31) za kot θ_1.

$$\theta_2 = arccos(\frac{8.50^2 + 27.72^2 - 17^2 - 13^2}{2 * 17 * 13}) \tag{4.32}$$

$$\theta_2 = 30^\circ \tag{4.33}$$

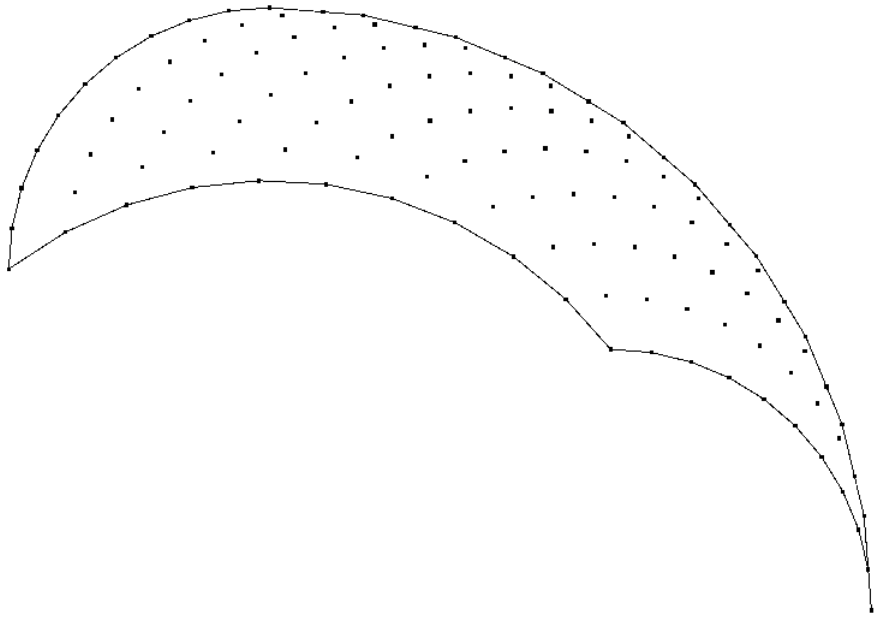
$$\theta_1 = arctan(\frac{27.72 * (17 + 13 * cos(30)) - 8.50(13 * sin(30))}{8.50(17 + 13 * cos(30)) + 27.72(13 * sin(30))}) \tag{4.34}$$

$$\theta_1 = 60^\circ \tag{4.35}$$

Kot vidimo, dobimo rešitev enako vhodnim kotom iz poglavja 4.1.1.

4.2 Konkavna ovojnica

Pri razvoju aplikacije za operacijski sistem Android smo razvili tudi modul, ki omogoča krmiljenje planarnega 2-DOF-robotskega manipulatorja. Uporabnik lahko z vlečenjem prsta po ekranu mobilne naprave izrisuje črte, katerih koordinate se prek IK preračunajo v konfiguracije kotov robotskega manipulatorja. Ker je delovni prostor robotske roke omejen, smo omejitve prenesli tudi v aplikacijo. Aplikacija uporabniku prikaže delovni prostor roke, znotraj katerega obstaja rešitev IK (Slika 4.7).



Slika 4.7: Ovojnica delovnega prostora in oblak točk

Oblika in velikost delovnega prostora robotske roke je odvisna od dolžine posameznega dela roke ter kota, ki ga servomotor v sklepu lahko doseže. Da bi grafično prikazali delovni prostor in se izognili točkam na ravnini, ki jih roka ne more doseči, smo izračunali ovojnico delovnega prostora. Postopek izračuna ovojnice delovnega prostora smo razdelil v dva koraka.

1. V prvem koraku s pomočje FK izračunamo oblak točk znotraj omejitev robotske roke. Več ko je točk v oblaku, natančneje lahko izračunamo

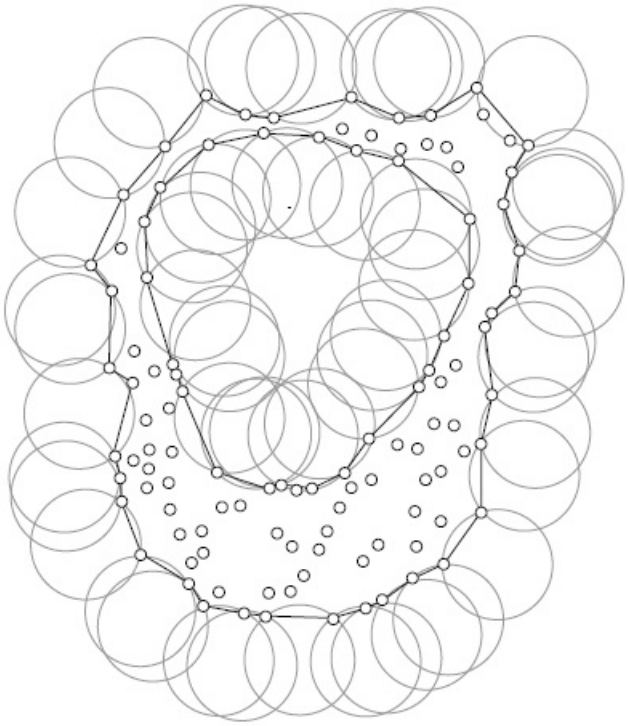
ovojnico delovnega prostora. Da bi bil celoten postopek izračuna ovojnice hitrejši, sklenemo kompromis in izračunamo le za približno 100 točk.

2. V drugem koraku izračunamo konkavno ovojnico nad oblakom točk iz prejšnjega koraka, ki jih metoda za izračun ovojnice dobi kot vhodni parameter. Izhod funkcije je seznam točk, ki predstavljajo ovojnico. Za izračun oblike oblaka točk oziroma konkavne ovojnice smo uporabili algoritem Alpha-shapes.

4.2.1 Algoritem Alpha-shapes

Algoritem Alpha-shapes je učinkovit pri izluščevanju konveksnih in konkavnih oblik predmetov in likov iz oblakov točk [2]. Algoritem se večinoma uporablja za upodabljanje 3D-modelov iz oblakov točk, ki jih posname sistem LIDAR (ang. *Light Detection and Ranging*). Tako lahko zajamejo površino mesta in ga upodobijo s 3D-modelom. V diplomskem delu smo algoritem uporabili za izračun konkavne ovojnice delovnega prostora robotskega manipulatorja.

Algoritem kot vhodne parametre prejme množico točk (oblak točk) S in α, ki predstavlja radij kroga (Slika 4.8) za detektiranje, ali sta dve točki na robu predmeta in predstavljata del ovojnice ali ne. S parametrom α lahko določamo, kako natančna je oblika predmeta, ki jo bo algoritem našel. Zato je primerna vrednost $\overline{L} < \alpha < 2\overline{L}$; $\overline{L}$ je povprečna razdalja med točkami v množici S [2].



Slika 4.8: Prikaz izrisanih krogov med dvema sosednjima točkama, ki določata rob oblaka točk

Algoritem

1. Iz množice točk S izberemo poljubno točko P_1.

2. Ustvarimo množico S_2, v kateri so vse točke iz množice S, katerih razdalja do $P_1 < 2 * \alpha$.

3. Iz množice S_2 izberemo poljubno točko P_2 in izračunamo središče točk P_1 in P_2, ki ga označimo s P_0. Koordinato x izračunamo z enačbo

(4.36), y izračunamo z enačbo (4.37).

$$P_{0x} = P_{1x} + \frac{(P_{2x} - P_{1x})}{2} + H(P_{2y} - P_{1y}) \tag{4.36}$$

$$P_{0y} = P_{1y} + \frac{(P_{2y} - P_{1y})}{2} + H(P_{1x} - P_{2x}) \tag{4.37}$$

$$H = \sqrt{\frac{\alpha^2}{S^2} - 0.25} \tag{4.38}$$

$$S = (P_{1x} - P_{2x})^2 + (P_{1y} - P_{2y})^2 \tag{4.39}$$

4. Preverimo, če sta točki P_1 in P_2 mejni točki, ki določata rob predmeta ali lika.

   - Če velja $\mathrm{d}(P_1, P_2) < \alpha$, potem sta točki P_1 in P_2 mejni točki in si shranimo stranico P_1P_2.
   - Če je $\mathrm{d}(P_1, P_2) >= \alpha$, prekinemo in se vrnemo na točko 3.

   To ponovimo za vse elemente množice S_2.

5. Za vse elemente množice S ponovimo 1. točko.

Poglavje 5

Razvoj in implementacija aplikacije

V tem poglavju je opisan razvoj aplikacije za operacijski sistem Android, ki smo jo razvili v programskem jeziku Java.

5.1 Struktura aplikacije in organizacija datotek

Ob kreiranju novega projekta za aplikacijo Android v IDE Android Studio nam program ustvari osnovno strukturo map in datotek, ki so potrebne za osnovno delovanje aplikacije. Med pomembnimi mapami sta *java* in *res*. V mapi *java* se nahaja naša koda aplikacije. V mapi *res* se nahajajo viri (ang. *resources*), kot so: zvoki, glasba, slike, ikone, itd. Poleg virov se v njej nahajajo tudi XML-datoteke, ki se uporabljajo za različne namene. Nameni uporabe XML-datotek v mapi *res*:

- za deklarativno programiranje grafičnih vmesnikov. XML-datoteko lahko napišemo ročno ali uporabimo grafični vmesnik za grajenje grafičnih vmesnikov. Grafični vmesnik se lahko generira tudi programsko;

- za stile komponent grafičnega vmesnika, pri čemer lahko definiramo popolnoma nov izgled komponent;
- za hranjenje nizov;
- za definiranje vsebine spustnih seznamov;
- za definiranje vsebine menijev;
- za definiranje nastavitev v aktivnosti, izpeljani iz abstraktnega razreda *PreferenceActivity* ...

5.1.1 Manifest

Najbolj pomembna datoteka je *AndroidManifest.xml*, saj se v njej nahaja definicija aplikacije. V manifestu se nahajajo podatki o ikoni in imenu aplikacije, tema, orientacija aktivnosti, pravice, minimalna in ciljna verzija Android SDK, verzija aplikacije, dodatne funkcionalnosti, aktivnosti itd. V IDE Android Studiu se lahko vsebino *AndroidManifest.xml* kombinira z datoteko *build.gradle*, ki jo uporablja sistem Gradle, opisan v poglavju 2.3.1. Zato smo minimalno in ciljno verzijo Android SDK definirali v datoteki *build.gradle* (Slika 5.1).

```
defaultConfig {
    applicationId "sonus.tempero.jarhead.temperosonusandroid"
    minSdkVersion 16
    targetSdkVersion 21
    versionCode 1
    versionName "1.0"
}
```

Slika 5.1: Minimalna in ciljna verzija Android SDK

Minimalna verzija Android SDK, ki smo jo podprli, je API Level 16, kar pomeni, da aplikacija lahko deluje na mobilnih napravh z verzijo operacijskega sistema od Androida 4.1 Jelly Bean naprej.

Ker aplikacija omogoča branje in pisanje v zunanjo shrambo, smo v datoteki *AndroidManifest.xml* dodali pravice (Slika 5.2).

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Slika 5.2: Pravice za branje in pisanje v zunanjo shrambo

5.2 Generiranje PPM-signala

Glavna funkcionalnost aplikacije je generiranje PPM-signala glede na konfiguracijo, ki jo definira uporabnik. Zato smo razvili razred *Emitter*, ki omogoča generiranje signala in predvajanje vsebine prek avdioizhoda na napravi.

Da lahko sproti generiramo in predvajamo generirano vsebino, smo uporabili razred *AudioTrack*, ki je namenjen upravljanju in predvajanju avdiovirov v javanskih aplikacijah [25].

Preden lahko ustvarimo objekt tipa *AudioTrack*, moramo izračunati velikost medpomnilnika (ang. *buffer*), ki ga potrebujemo za generiranje enega PPM-signala. Pri izračunu velikosti medpomnilnika se upoštevajo vse nastavitve uporabnika, ki so podane kot argumenti v konstruktorju, razen dolžine začetnega premora. Ta se pri izračunu izpusti, ker se velikost medpomnilnika ne more spremeniti, ko je objekt tipa *AudioTrack* že ustvarjen. Ker dolžini začetnega premora in dolžina enega PPM-signala nista enaki, bi morali velikost medpomnilnika spremeniti. Zato se začetni premor generira ločeno.

Pri izračunu dolžine medpomnilnika (Koda 5.1) se najprej izračuna maksimalno trajanje enega PPM-signala. Potem se čas trajanja pretvori v število vzorcev. Pri izračunu velikosti medpomnilnika moramo paziti, da je velikost medpomnilnika sodo število, saj sicer pri predvajanju vsebine medpomnilnika pride do proženja izjeme. V primeru, da uporabljamo stereo konfiguracijo avdiokanala, moramo velikost medpomnilnika pomnožiti z 2, ker vsak avdiokanal potrebuje svojo polovico medpomnilnika.

Koda 5.1: Izračun velikosti medpomnilnika

```
1  public int getBufferSize(int channelConfig) {
2      float size = 0;
3      size += syncPause;
4      size += (channelNum+1)*pauseLength;
5      for(int i=0;i<chPWM.length;i++) {
6          size+= chPWM[i].second;
7      }
8
9      int bufSize = (int) ((frequency / 1000f) * size);
10     if(bufSize%2 != 0)
11         bufSize += 1;
12     this.multiply = 1;
13
14     if(channelConfig == AudioFormat.CHANNEL_OUT_STEREO) {
15         bufSize*=2;
16         this.multiply = 2;
17     }
18     return bufSize;
19 }
```

Ko imamo izračunano velikost medpomnilnika, lahko kreiramo objekt tipa *AudioTrack*. Glavne lastnosti, ki smo jih nastavili objektu, so:

- vzorčenje smo nastavili na 44100 Hz;
- deluje naj v tokovnem načinu (ang. *Stream mode*), da lahko sproti generiramo podatke v medpomnilniku in jih predvajamo;
- format predstavitve podatkov smo nastavili na PCM_16BIT (ang. *Pulse-code modulation*), kar pomeni, da je vsak podatek zapisan z dvema bajtoma. Format zapisa moramo upoštevati tudi pri izračunu velikosti medpomnilnika. V našem primeru za hranjenje medpomnilnika uporabljamo tabelo tipa *short*, ki je v spominu predstavljena z dvema bajtoma, zato nam ni bilo treba posebej paziti pri izračunu. Če bi medpomnilnik hranili v podatkovnem tipu *byte*, bi morali velikost medpomnilnika pomnožiti z 2.

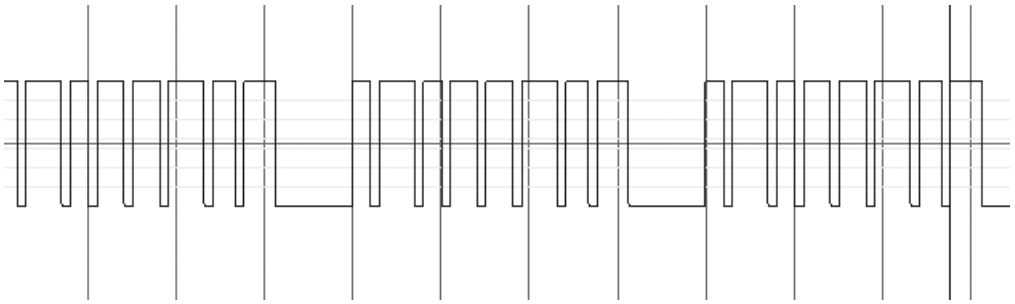
Pri frekvenci vzorčenja 44100 Hz imamo na voljo 44 vzorcev v časovnem okvirju 1 ms. Tako moramo na 44 vzorcev razdeliti celotni delovni kot servomotorja, ki ga lahko pokrije. V primerih, ko ima servomotor velik delovni kot, na primer 180°, je resolucija kota zelo slaba, saj en vzorec predstavlja kar 4°. Resolucijo kota lahko izboljšamo, tako da uporabljamo servomotorje z manjšim delovnim kotom.

Zdaj je objekt tipa *AudioTrack* pripravljen za uporabo in lahko začnemo z generiranjem PPM-signala. To storimo tako, da kličemo metodo *emit*, kateri se kot vhodni argument pošlje tabelo tipa *float* z vrednostmi kotov za vsak aktiven izhod na vezju. Število elementov tabele argumenta mora biti enako, kot smo ga podali v konstruktorju razreda *Emitter*. Koti, ki se pošljejo v metodo, so v stopinjah. V metodi se za vsak kot v tabeli izračuna čas trajanja pulza, ki se pretvori v število vzorcev medpomnilnika, potem se pulz zapiše v medpomnilnik (Koda 5.2). Med vsak pulz kota se vstavi kratek premor dolžine 0.3 ms, ki ločuje PWM-signale.

Koda 5.2: Generiranje PPM-signala za Mono

```
1  public boolean emit(float[] angles) {
2          if(angles == null
3                  || this.channelNum != angles.length) {
4                  return false;
5          }
6
7      //Init offset of buffer
8      Arrays.fill(this.buffer, maxValue);
9      int offset = 0;
10     int addOffset = (int) msToUnit(pauseLength);
11     //Hardware counter reset signal
12     this.writeToBuffer(minValue, offset, addOffset);
13     offset += addOffset;
14
15     //Angles
16     for (int i = 0; i < angles.length; i++) {
17         int angleToTime = (int) msToUnit(angleToMs(angles[i], i));
18         int pause = (int) msToUnit(pauseLength);
19
20         this.writeToBuffer(maxValue, offset, angleToTime);
21         offset += angleToTime;
22
23         this.writeToBuffer(minValue, offset, pause);
24         offset += pause;
25     }
26     this.writeToBuffer(minValue, offset, this.buffer.length - offset);
27     return this.play();
28 }
```

Ko se medpomnilnik napolni, se ga prepiše v *AudioTrack* in predvaja. Na sliki 5.3 je primer generiranega izhodnega signala.



Slika 5.3: Generiran PPM-signal za 8 servomotorjev

Vsak modul, ki uporablja razred *Emitter*, poleg glavne niti (ang. *Thread*) uporablja še dodatno nit, v kateri se v neskončni zanki kliče metoda *emit* z vrednostmi kotov.

5.3 Aktivnosti

Pri razvoju aplikacij za operacijski sistem Android se programiranje začne z razširitvijo razreda *Activity* ali izpeljanih razredov. Aktivnost je vrsta razreda, ki skrbi za kreiranje okna, v katerega lahko vstavimo grafični vmesnik. V diplomski nalogi smo razrede aktivnosti dedovali iz razredov *ActionBarActivity* in *PreferenceActivity*. Dedovanje iz razreda *ActionBarActivity* smo uporabili pri vseh aktivnostih, ki uporabljajo grafični vmesnik po meri in za katere želimo, da imajo akcijsko vrstico, v katero lahko vstavimo gumb za nastavitve. Dedovanje iz razreda *PreferenceActivity* smo uporabili pri aktivnostih z nastavitvami.

5.3.1 Aktivnost glavnega menija

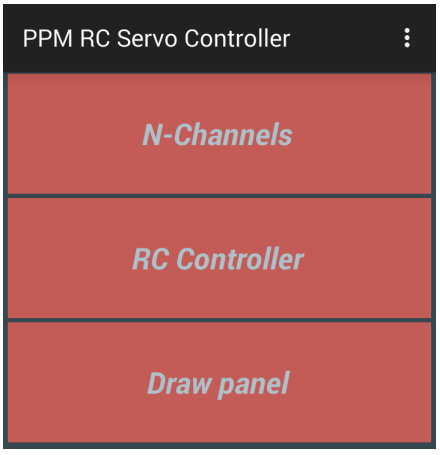
Aktivnost glavnega menija se zažene kot začetna aktivnost aplikacije. To smo definirali v datoteki *AndroidManifest.xml* (Koda 5.3).

Koda 5.3: Definiranje začetne aktivnosti v *AndroidManifest.xml*

```
<activity
                android:name=".activity.MainMenuListActivity"
                android:configChanges="orientation|keyboardHidden|screenSize"
                android:label="@string/app_name"
                android:screenOrientation="portrait"
                android:theme="@style/AppTheme" >
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

Glavni meni je sestavljen iz seznama gumbov, ki vodijo do različnih modulov aplikacije (Slika 5.4). Da lahko prikažemo gumbe v seznamu (ang. *ListView*), smo implementirali *ArrayAdapter* po meri. Aktivnost glavnega menija vsebuje splošne nastavitve generatorja PPM-signala, kjer lahko nastavimo:

- invertiranje signala,
- takojšnje generiranje PPM-signala ob zagonu aktivnosti modula,
- začetni premor pri generiranju PPM-signala in
- dolžino začetnega premora pri generiranju PPM-signala.

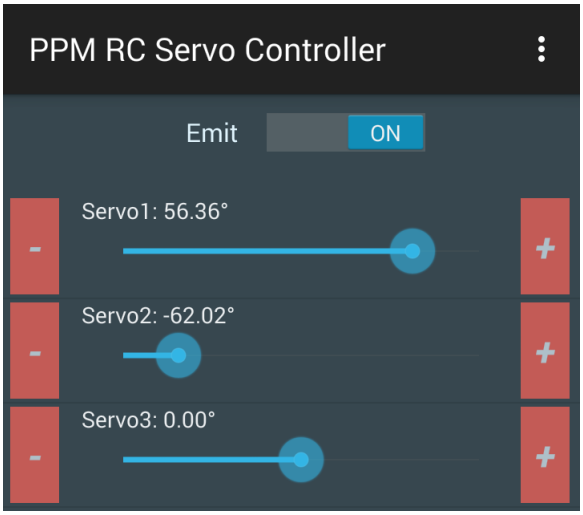


Slika 5.4: Glavni meni

5.3.2 Aktivnost za upravljanje z N-servomotorji

Prvi modul, ki smo ga razvili, je modul za upravljanje z N-servomotorji. Za krmiljenje PWM-signala na vsakem izhodu vezja kot drsnik uporabljamo kontrolo za iskanje (ang. *SeekBar*). Prav tako kot pri aktivnostih glavnega menija tudi pri tej aktivnosti uporabljamo seznam za prikazovanje kontrol posameznega izhoda na vezju in stikalo za vklop ali izklop generatorja PPM-signala (Slika 5.5). Vsaka vrstica seznama vsebuje drsnik za določanje kota, dva gumba za spreminjanje kota po korakih v pozitivno ali negativno smer in besedilo, ki prikazuje vrstni red izhoda na vezju in trenutni kot. V tem modulu lahko nastavljamo različne splošne nastavitve za generiranje PPM-signala in nastavitve za vsak posamezni izhod posebej. Nastavljamo lahko:

- dolžino sinhronizacijskega premora,
- korak gumbov za spreminjanje kota,
- število avdiokanalov,
- število aktivnih izhodov na vezju,
- maksimalno in minimalno širino posameznega PWM-pulza in
- obračanje signala.



Slika 5.5: Modul za upravljanje z N-servomotorji s 3 aktivnimi izhodi

5.3.3 Aktivnost za upravljanje s servomotorji z virtualnima igralnima palicama

Ta modul je zelo podoben prvemu. Razlikuje se v tem, da za spreminjanje kotov uporablja dve kontroli z virtualnima palicama in da ne more spreminjati števila aktivnih izhodov na vezju; uporablja prvih 8 izhodov (Slika 5.6). Vse nastavitve, ki jih definiramo v prvem modulu, se uporabljajo tudi v tem modulu. Dodatne nastavitve, ki jih lahko nastavljamo, so:

- vklop besedila z vrednostmi koordinat,
- vklop dodatnih 4 gumbov, da lahko koristimo še preostale 4 kanale,
- zaklep osi igralne palice in
- vklop vzmeti v igralni palici, da se ob sprostitvi dotika vrne v izhodišče.

Ker Android SDK ne ponuja kontrole, ki bi jo lahko uporabili kot virtualno igralno palico, smo implementirali svojo kontrolo, ki programsko izriše igralno palico. Zato smo razvili razred *Joystick*, ki skrbi za izris igralne palice (manjši krog na sliki 5.6) in izračuna odmik igralne palice od izhodišča. Da

Slika 5.6: Modul za upravljanje s servomotorji z virtualnima igralnima palicama

lahko rišemo na obstoječo kontrolo, moramo dodati platno (ang. *Canvas*), na katerem lahko izrisujemo (Koda 5.4).

Koda 5.4: Dodajanje platna na obstoječo kontrolo grafičnega vmesnika

```
1  private ViewGroup mLayout;
2  private DrawCanvas canvas;
3
4  private void draw() {
5          try {
6                  mLayout.removeView(canvas);
7          }
8          catch (Exception e) {
9                  e.printStackTrace();
10         }
11         mLayout.addView(canvas);
12 }
13
14 private class DrawCanvas extends View {...}
```

5.3.4 Risalna plošča

Tretji modul, ki smo ga razvili, je risalna plošča. Ta modul omogoča krmiljenje planarnega 2-DOF-manipulatorja. Robotska roka, ki jo krmili modul, uporablja 3 servomotorje. Dva motorja uporablja za sklepe in tretji motor za vzdigovanje peresa na vrhu roke. Modul omogoča uporabniku, da lahko po ekranu mobilne naprave s prstom riše črte, ki se shranijo kot koordinate točk. S pomočjo IK se jih pretvori v kote robotskega manipulatorja in z razredom *Emitter* generira PPM-signal.

Da lahko rišemo črte, moramo najprej označiti prostor dosega robotske roke. Delovni prostor roke najdemo tako, da s FK generiramo oblak točk, nad katerim pozneje poiščemo konkavno ovojnico z algoritmom Alpha-shapes. Teorija izračuna konkavne ovojnice delovnega prostora je opisana v poglavju 4.2. Preden lahko začnemo z izračunavanjem delovnega prostora, moramo dodati lastnosti roke, ki se v modul vnesejo z nastavitvami (Slika 5.7):

- dolžine posameznega dela roke,
- minimalnega želenega kota motorja,
- maksimalnega želenega kota motorja in
- maksimalnega kota motorja po dokumentaciji.

Slika 5.7: Aktivnost z nastavitvami

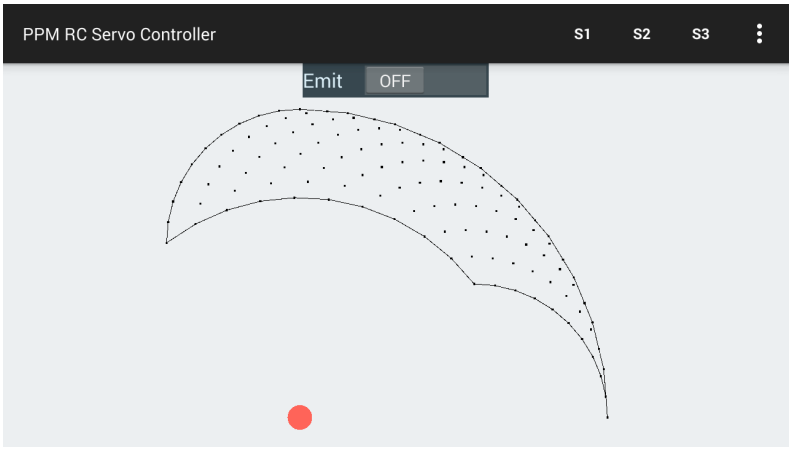
Modul omogoča tudi nastavljanje parametra α, število generiranih točk v oblaku in hitrost izrisovanja robotskega manipulatorja.

Modul z risalno ploščo ob zagonu izračuna in izriše delovni prostor robotskega manipulatorja. Risalna plošča omogoča premikanje in skaliranje površine z običajnimi kretnjami. Za omogočanje skaliranja je treba implementirati vmesnik *ScaleGestureDetector.OnScaleGestureListener* (Koda 5.5), ki zna sam zaznati kretnjo skaliranja in proži dogodek, na katerega se je potrebno registrirati.

Koda 5.5: Skaliranje delovnega prostora

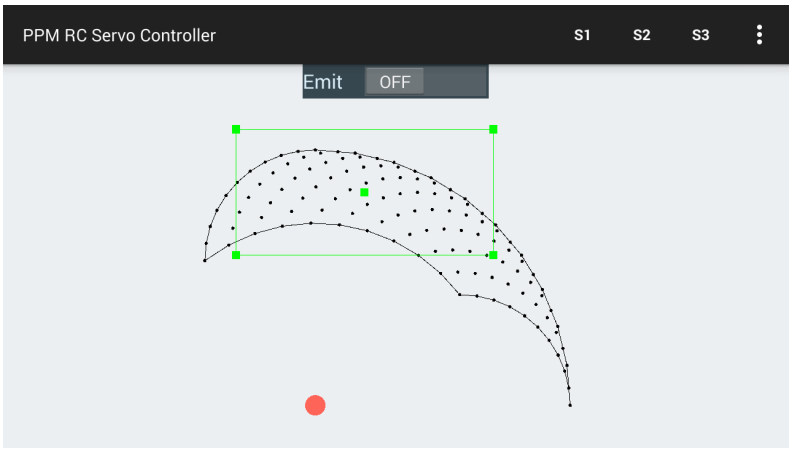
```
1  public boolean onScale(ScaleGestureDetector detector) {
2          int workflowStep = 1; workflowStep = Preferences.drawpanelStep(prefManager);
3          if(workflowStep <= 1) {
4                  float s = scale;
5                  s *= detector.getScaleFactor();
6                  scale = s;
7          } if(workflowStep == 2 && workspace != null) {
8                  float s = workspace.getScale();
9                  s *= detector.getScaleFactor();
10                 workspace.setScale(s);
11         }
12         invalidate();
13         return true;
14 }
```

Modul vsebuje 3 korake, s katerimi nastavimo okno v delovni prostor robotskega manipulatorja. V vrstici z akcijami (zgornji desni kot) so trije gumbi: S1, S2 in S3 (Slika 5.8). Vsak predstavlja en korak za namestitev delovnega prostora. V koraku S1, kjer se modul tudi naloži, lahko s kretnjami premikamo in skaliramo celotni delovni prostor.



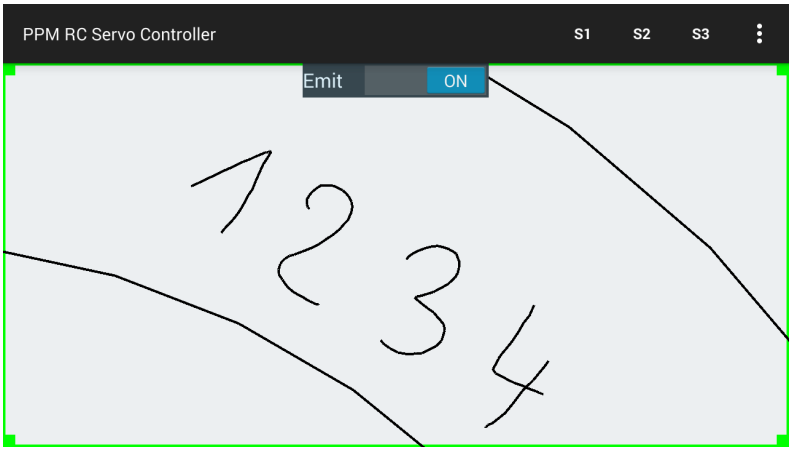
Slika 5.8: Modul z risalno ploščo v stanju S1

V koraku S2 se pojavi zelen okvir, ki ima stranice v takem razmerju kot stranice ekrana naprave, in predstavlja okno, v katerem bomo lahko risali črte (Slika 5.9). To okno lahko v koraku S2 premikamo in skaliramo.



Slika 5.9: Modul z risalno ploščo v stanju S2

V koraku S3 se zelen okvir, ki predstavlja naše okno, skalira na velikost ekrana in zapolni celoten ekran (Slika 5.10). Zdaj ob dotiku ekrana lahko vlečemo črte. Med koraki lahko prestopamo v poljubnem vrstnem redu in tako premaknemo okno v delovnem prostoru tudi na druga območja delovnega prostora robotskega manipulatorja.



Slika 5.10: Modul z risalno ploščo v stanju S3

Pri implementaciji tranformacij za premikanje in skaliranje delovnega prostora in okna na delovni prostor smo uporabili knjižnico vec-math, ki vsebuje razrede za delo z matrikami in vektorji. Za računanje transformacij smo implementirali statični razred *Transformation*, ki implementira metodi za premikanje in skaliranje točk (Koda 5.6).

Koda 5.6: Matriki za premikanje in skaliranje točk

```
1  public static Vector2f translate(Vector2f vector, float dx, float dy, float dz) {
2          Matrix4f matrix = new Matrix4f();
3          matrix.setRow(0, new float[]{1,0,0,dx});
4          matrix.setRow(1, new float[]{0, 1, 0, dy});
5          matrix.setRow(2, new float[]{0,0,1,dz});
6          matrix.setRow(3, new float[]{0,0,0,1});
7          return mulVector(matrix, vector);
8  }
9
10 public static Vector2f scale(Vector2f vector, float sx, float sy, float sz) {
11         Matrix4f matrix = new Matrix4f();
12         matrix.setRow(0, new float[]{sx,0,0,0});
13         matrix.setRow(1, new float[]{0,sy,0,0});
14         matrix.setRow(2, new float[]{0,0,sz,0});
15         matrix.setRow(3, new float[]{0,0,0,1});
16         return mulVector(matrix, vector);
17 }
```

Implementacija IK

Za reševanje problema IK smo implementirali razred *Inverse* (Koda 5.7), ki implementira geometrijski pristop, opisan v poglavju 4.1.2. Razred se uporablja tako, da se v konstruktorju poda dolžine delov robotskega manipulatorja. Pred izračunom IK se nastavi ciljno pozicijo x in y in potem kliče metodo *compute*, ki izračuna kota za sklepa.

Koda 5.7: Implementacija IK

```
1  public void compute() {
2          double distanceSquare = Math.pow(position.length(), 2);
3          double imLength = 2*this.armLength1*this.armLength2;
4          double cAngle2 = 0;
5          double sAngle2 = 0;
6          if(imLength > epsilon) {
7                  cAngle2 = (
8                          distanceSquare
9                          - Math.pow(this.armLength1,2)
10                         - Math.pow(this.armLength2,2)
11                         ) / imLength;
12                 cAngle2 = Math.max(-1, Math.min(1, cAngle2));
13                 this.angle2 = Math.acos(cAngle2);
14                 sAngle2 = Math.sin(this.angle2);
15         }
16         else {
17                 this.angle2 = 0.0f; cAngle2 = 1.0f; sAngle2 = 0.0f;
18         }
19         double A = this.armLength1 + this.armLength2*cAngle2;
20         double B = this.armLength2*sAngle2;
21         double tanY = this.position.y*A - this.position.x*B;
22         double tanX = this.position.x*A + this.position.y*B;
23         this.angle1 = Math.atan2(tanY, tanX);
24 }
```

Implementacija Alpha-shapes

Za izračun konkavne ovojnice okoli oblaka točk delovnega prostora smo implementirali algoritem Alpha-shape, opisan v poglavju 4.2. Zato smo razvili razred *Concave*, ki implementira opisan algoritem (Koda A.1).

Shranjevanje in nalaganje datoteke

Modul z risalno ploščo omogoča nalaganje in trajno hranjenje črt, za kar uporabljamo datoteko formata JSON. Ob shranjevanju v datoteko zapišemo časovni žig izvoza in za vsako črto v datoteko zapišemo tabelo x in y koordinate (Koda 5.8). Pri tem uporabljamo knjižnico json-simple, s katero

kodiramo podatke za hranjenje in dekodiramo vsebino JSON-datoteke pri nalaganju.

Koda 5.8: Izvoz izrisanih črt v datoteko formata JSON

```
1  public boolean exportLines(String fileName){
2          if(this.positions != null){
3                  try{
4                          JSONObject jsonObject = new JSONObject();
5                          jsonObject.put("exported",(new java.util.Date()).toString());
6                          JSONArray lines = new JSONArray();
7                          for(List<Vector2f> line : this.positions){
8                                  JSONArray jsonArray = new JSONArray();
9                                  for(Vector2f vertex : line){
10                                         JSONObject vertexObj = new JSONObject();
11                                         vertexObj.put("x", String.format("%.2f", vertex.x));
12                                         vertexObj.put("y", String.format("%.2f", vertex.y));
13                                         jsonArray.add(vertexObj);
14                                 }
15                                 lines.add(jsonArray);
16                         }
17                         jsonObject.put("lines", lines);
18                         sonus.tempero.jarhead.temperosonusandroid.utility.File
19                         .WriteToFile(File.drawingPath,
20                                                 fileName+".json",
21                                                 jsonObject.toString(), this.activity);
22                         return true;
23                 }
24                 catch (FileNotFoundException e) { e.printStackTrace();
25                         Log.e("SaveData", "Can_not_write_to_the_file!"); }
26                 catch (IOException e){ e.printStackTrace(); }
27         }
28         return false;
29 }
```

Generiranje in predvajanje PPM-signala za risalno ploščo

Za generiranje in predvajanje PPM-signala v modulu risalne plošče smo uporabili že obstoječ razred *Emitter*. Vendar smo morali dodati funkcionalnost, ki omogoča pošiljanje koordinat črt in ukaza za vzdigovanje peresa, v metodo *emit*, razred *Emitter*, glede na zmogljivosti servomotorjev (nastavljivo v nastavitvah). Tako smo razvili razred *Timer* (Koda 5.9), s katerim v novi niti ustvarimo števec. Z njim dosežemo, da se nova koordinata v metodo *emit* pošlje na vsake N ms, čeprav se metoda *emit* kliče v neskončni zanki, brez premora, da ustvarja nenehno prisotnost PPM-signala.

Koda 5.9: Nastavitev števca

```
1  public void setTimer(long time) {
2          try {
3                  final long delayTime = time;
4                  this.elapsed = false;
5                  Thread thread = new Thread(new Runnable() {
6                          @Override
7                          public void run() {
8                                  try { Thread.sleep(delayTime); }
9                                  catch (InterruptedException e) { e.printStackTrace(); }
10                                 elapsed = true;
11                         }
12                 });
13                 thread.start();
14         }
15         catch (Exception ex) { ex.printStackTrace(); }
16 }
```

5.3.5 Aktivnosti z nastavitvami

Pri aktivnostih za nastavitve smo se odločili za razširitev razreda *SharedPreferences*, saj nam omogoči avtomatsko shranjevanje in nalaganje nastavitve iz shrambe skupnih nastavitev (ang. *Shared preferences*).

Ob zagonu aktivnosti z nastavitvami ustvarimo nov fragment, v katerem se bo nahajal seznam nastavitev. Vsebino na novo kreiranega fragmenta zamenjamo s trenutno vsebino aktivnosti (Koda 5.10).

Koda 5.10: Zamenjava vsebine aktivnosti s fragmentom

```
1  PreferenceFragmentDrawPanel prefs;
2
3  @Override
4  public void onCreate(Bundle savedInstanceState) {
5          super.onCreate(savedInstanceState);
6          getWindow().setFlags(
7                  WindowManager.LayoutParams.FLAG_FULLSCREEN,
8                  WindowManager.LayoutParams.FLAG_FULLSCREEN);
9          getWindow().addFlags(
10                 WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
11
12         if(prefs == null) {
13                 prefs = new PreferenceFragmentDrawPanel();
14         }
15
16         getFragmentManager()
17                 .beginTransaction()
18                 .replace(
19                 android.R.id.content, prefs).commit();
20 }
```

V fragmentu naložimo definicijo nastavitev iz XML-datotek (Koda 5.11). V XML-datoteki se nahajajo kategorije nastavitve, po katerih so nastavitve

združene. Znotraj kategorij so definicije nastavitev, ki so lahko predstavljene z različnimi tipi gradnikov grafičnega vmesnika (Koda 5.12). V diplomski nalogi smo uporabili tipe:

- **ListPreference** je spustni seznam, katerega vsebino lahko definiramo z XML-dokumentom. Ta tip nastavitve uporabljamo za nastavitve, ki imajo vnaprej določene izbire.
- **CheckBoxPreference** je potrditveno polje. Ta tip nastavitve uporabljamo za nastavitve, ki imajo lahko samo dve možni izbiri.
- **EditTextPreference** je okno z vnosnim poljem, kateremu lahko definiramo tip. Ta tip nastavitve uporabljamo za nastavitve, ki imajo številske vrednosti.
- **Preference** je tip nastavitve, ki mu podamo grafični vmesnik po meri, definiran v XML-datoteki. Ta tip nastavitve uporabljamo za vstavitev gumba v seznam nastavitve, ki odpre novo aktivnost z nastavitvami.

Koda 5.11: Nalaganje vsebine XML v aktivnost nastavitev

```
1  @Override
2  public void onCreate(final Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          addPreferencesFromResource(
5                  R.xml.preferences_drawpanel);
6
7          Preferences.displaySelectedValues(
8                          this.getActivity(),
9                          this);
10 }
```

Koda 5.12: Primer vsebine XML-datotek za nastavitve

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3      <PreferenceCategory
4              android:title="@string/draw_panel_preferences_workspace_category">
5          <EditTextPreference
6                  android:title="@string/draw_panel_preferences_workspace_alpha_title"
7                  android:key="alpha"
8                  android:summary="@string/draw_panel_preferences_workspace_alpha_summary"
9                  android:defaultValue="75"
10                 android:inputType="numberDecimal"
11                 ></EditTextPreference>
12         </PreferenceCategory>
13 </PreferenceScreen>
```

Za pridobivanje vrednosti nastavitev smo implementirali statični razred *Preferences*, ki vsebuje metode za branje iz skupnih nastavitev. Za vsako nastavitev obstaja metoda, ki prebere in vrne vrednost nastavitve. Če nastavitev še nikoli ni bila shranjena in ne obstaja v skupnih nastavitvah, metoda vrne privzeto vrednost (Koda 5.13).

Koda 5.13: Primer implementacije metode za pridobivanje vrednosti o minimalni in maksimalni širini PWM-pulza

```
1  public static Pair<Float, Float> pwmWidth(SharedPreferences prefManager, int index) {
2          Pair<Float, Float> opt = new Pair<>(1.0f, 2.0f);
3          if(prefManager != null){
4                  int option = Integer.valueOf(
5                          prefManager.getString(
6                                  String.format("signal_channel_pwm_width_%d", index), "6"));
7                  switch(option){
8                          case 1: opt = new Pair<>(0.5f, 2.5f); break;
9                          case 2: opt = new Pair<>(0.6f, 2.4f); break;
10                         case 3: opt = new Pair<>(0.7f, 2.3f); break;
11                         case 4: opt = new Pair<>(0.8f, 2.2f); break;
12                         case 5: opt = new Pair<>(0.9f, 2.1f); break;
13                         case 6: default: opt = new Pair<>(1.0f, 2.0f); break;
14                 }
15         }
16         return opt;
17 }
```

5.4 Testiranje

Pri razvoju programske opreme je zelo pomembno sprotno testiranje, saj tako zmanjšamo število napak. S sprotnim testiranjem funkcionalnosti si olajšamo nadaljnji razvoj, saj ob detekciji novih napak lahko izločimo možnosti napak v že testiranih funkcionalnostih in tako enostavneje lociramo novo napako. V praksi ni vedno tako enostavno, kot se morda sliši. Pri razvoju aplikacije smo testirali na mobilnih napravah:

- Samsung S3 Neo GT-i9301I (Android 4.4.2 KitKat),
- LG Nexus 5 (Android 5.1.1 Lollipop) in
- LG G2 (Android 4.2.2 Jelly Bean).

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo razvili dekodirno vezje in aplikacijo za mobilni operacijski sistem Android, ki generira PPM-signal. Cilj diplomskega dela je bil razviti mobilno aplikacijo, ki omogoča krmiljenje servomotorjev prek generiranega PPM-signala, ki se prenaša skozi avdiokanal v dekodirno vezje. Sprva smo nameravali uporabiti samo en avdiokanal, s katerim lahko krmilimo do 8 servomotorjev hkrati. Vendar smo pozneje idejo razširili, da lahko uporabljamo 2 avdiokanala in tako krmilimo do 16 servomotorjev hkrati. Ker smo dekodirno vezje razvili z 9 izhodi za servomotorje, lahko zdaj dejansko krmilimo do 18 servomotorjev hkrati. Ker je bila aplikacija razvita za primer didaktičnega pripomočka, smo v njej izpostavili skoraj vse nastavljive parametre v nastavitvah, tako da ni vezana na strogo določeno strojno opremo in ponuja možnost za različne konfiguracije.

Kot primer uporabe aplikacije je bil razvit 3. modul, v katerem lahko z vlečenjem črt na ekranu mobilne naprave krmilimo planarni 2-DOF-manipulator. Za testiranje 3. modula smo razvili tudi robotsko roko, vendar zaradi odpovedi dveh servomotorjev nismo uspeli do konca preveriti, kako se aplikacija obnaša s pravo robotsko roko.

Pri razvoju aplikacije nismo imeli večjih težav, čeprav smo se prvič srečali z razvojem za operacijski sistem Android. V veliko pomoč nam je bila uradna dokumentacija operacijskega sistem Android, ki je zelo dobro dokumentirana.

Pri razvoju dekodirnega vezja smo imeli nekaj težav s šumom v ojačanem signalu, v primeru, ko smo generirali PPM-signal za manj kot 8 aktivnih izhodov. Vendar smo ugotovili, da je bil problem v enem izmed čipov, ki je povzročal težave.

Pri izdelavi diplomskega dela smo pridobili veliko novega znanja in izkušenj. V razvoj aplikacije je potrebno vložiti še nekaj dela, jo optimizirati, razširiti s še kakšno novo funkcionalnostjo, izboljšati grafični vmesnik in prenesti aplikacijo na operacijski sistem iOS.

Dodatek A

Dodatek - Implementacija algoritma Alpha-shapes

Koda A.1: Implementacija algoritma Alpha-shapes

```
1  public class Concave
2  {
3      private Concave()
4      {
5
6      }
7
8      private static Vector2f drawCircle(Vector2f p1, Vector2f p2, double alpha)
9      {
10         Vector2f p3 = new Vector2f();
11
12         double s2 = Math.pow((p1.x - p2.x), 2) + Math.pow((p1.y - p2.y), 2);
13         double h = Math.sqrt(alpha*alpha/s2 - 0.25);
14
15         p3.x = (float) (p1.x + 0.5*(p2.x - p1.x) + h*(p2.y - p1.y));
16         p3.y = (float) (p1.y + 0.5*(p2.y - p1.y) + h*(p1.x - p2.x));
17
18         return p3;
19     }
20
21     public static List<Vector2f[]> compute(List<Vector2f> points, double alpha)
22     {
23         if(points != null)
24         {
25             List<Vector2f> pointsHull = getHull(points, alpha);
26
27             List<Vector2f[]> pts = new ArrayList<>();
28             for(int i=0;i<pointsHull.size();i+=2)
29             {
30                 Vector2f[] c = new Vector2f[2];
31
32                 c[0] = new Vector2f();
33                 c[0].x = pointsHull.get(i).x;
34                 c[0].y = pointsHull.get(i).y;
35
```

```
36                  c[1] = new Vector2f();
37                  c[1].x = pointsHull.get(i+1).x;
38                  c[1].y = pointsHull.get(i+1).y;
39
40                  pts.add(c);
41              }
42
43              return pts;
44          }
45
46          return null;
47      }
48
49      private static List<Vector2f> findPointsInRange(Vector2f refPoint,
50                                                      List<Vector2f> points, double maxDistance)
51      {
52          List<Vector2f> inRange = new ArrayList<>();
53          for(int i=0;i<points.size();i++)
54          {
55              double d = Math.sqrt(
56                                  (refPoint.x - points.get(i).x) * (refPoint.x - points.get(i).x)
57                                + (refPoint.y - points.get(i).y) * (refPoint.y - points.get(i).y));
58              if(d < maxDistance){
59                  inRange.add(points.get(i));
60              }
61          }
62
63          return inRange;
64      }
65
66      private static List<Vector2f> getHull(List<Vector2f> points, double alpha)
67      {
68          double alpha2 = 2*alpha;
69          List<Vector2f> edges = new ArrayList<>();
70          List<Vector2f> ppoints = new ArrayList<>();
71          ppoints.addAll(points);
72
73          while(!ppoints.isEmpty())
74          {
75              Vector2f p1 = ppoints.remove(0);
76              List<Vector2f> inRange = findPointsInRange(p1, points, alpha2);
77
78              if(inRange.size() < 2)
79              {
80                  break;
81              }
82              inRange.remove(p1);
83
84              boolean alreadyChecked[] = new boolean[inRange.size()];
85              Arrays.fill(alreadyChecked, false);
86              int ppointsCounter = inRange.size();
87              int pointIndex = 0;
88              while(ppointsCounter > 0)
89              {
90                  Vector2f p2;
91                  while(true)
92                  {
93                      if (!alreadyChecked[pointIndex])
94                      {
95                          p2 = inRange.get(pointIndex);
96                          alreadyChecked[pointIndex] = true;
97                          ppointsCounter--;
98                          pointIndex++;
99                          break;
```

```
100                     }
101                 }
102
103                 Vector2f p0 = drawCircle(p1, p2, alpha);
104                 boolean boundary = true;
105                 for (Vector2f vp : inRange)
106                 {
107                     if (vp != p2)
108                     {
109                         double len = Math.sqrt(
110                                 (p0.x - vp.x) * (p0.x - vp.x)
111                                 + (p0.y - vp.y) * (p0.y - vp.y));
112                         if (len <= alpha)
113                         {
114                             boundary = false;
115                             break;
116                         }
117                     }
118                 }
119
120                 if (boundary)
121                 {
122                     edges.add(p1);
123                     edges.add(p2);
124                 }
125             }
126         }
127
128         return edges;
129     }
130 }
```

Literatura

[1] Klemen Kozjek and Aleš Jaklič. Didactic Tool: RC Servo Controller for Educational Robotics. In *Proceedings of the 24nd International Electrotechnical and Computer Science Conference*, 2015

[2] Wei Shen, Jin Zhang, and Feng Yuan. A new algorithm of building boundary extraction based on lidar data. In *Geoinformatics, 2011 19th International Conference on*, pages 1–4. IEEE, 2011.

[3] S. Kucuk and Z. Bingul. *Robot Kinematics: Forward and Inverse Kinematics*. INTECH Open Access Publisher, 2006.

[4] Tadej Bajd in Ivan Bratko, Robotika in umetna intelegenca, Slovenska matica, 2014.

[5] C. Cannam, C. Landone, and M. Sandler. Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files. In *Proceedings of the ACM Multimedia 2010 International Conference*, pages 1467–1468, Firenze, Italy, October 2010.

[6] Wikipedia, Prostostne stopnje. Dosegljivo: https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics). [Dostopano 9. 8. 2015].

[7] Strojni dekodirnik. Dosegljivo: http://sm0vpo.altervista.org/use/rc-prop.htm. [Dostopano 9. 8.

[8] Dekodirnik. Dosegljivo:
http://myweb.tiscali.co.uk/norcimradiocontrol/Radio6.htm. [Dostopano 9. 8.

[9] Wikipedia, Pulzno-pozicijska modulacija. Dosegljivo:
https://en.wikipedia.org/wiki/Pulse-position_modulation. [Dostopano 9. 8. 2015].

[10] Wikipedia, Širinsko-pulzna modulacija. Dosegljivo:
https://en.wikipedia.org/wiki/Pulse-width_modulation. [Dostopano 9. 8. 2015].

[11] Programski jezik C++. Dosegljivo:
http://www.cplusplus.com. [Dostopano 9. 8. 2015].

[12] Wikipedia, Datotečni format za izmenjavo virov. Dosegljivo:
https://en.wikipedia.org/wiki/Resource_Interchange_File_Format. [Dostopano 9. 8. 2015].

[13] CMake. Dosegljivo:
http://www.cmake.org/. [Dostopano 9. 8. 2015].

[14] Wikipedia, Navidezni stroj Dalvik. Dosegljivo:
https://en.wikipedia.org/wiki/Dalvik_(software). [Dostopano 9. 8. 2015].

[15] Wikipedia, Zgodovina Android verzij. Dosegljivo:
https://en.wikipedia.org/wiki/Android_version_history. [Dostopano 9. 8. 2015].

[16] Wikipedia, Operacijski sistem Android. Dosegljivo:
https://en.wikipedia.org/wiki/Android_(operating_system). [Dostopano 9. 8. 2015].

[17] Git. Dosegljivo: https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control. [Dostopano 9. 8. 2015].

[18] Android SDK. Dosegljivo: http://developer.android.com/tools/help/index.html. [Dostopano 9. 8. 2015].

[19] Wikipedia, Android SDK. Dosegljivo: https://en.wikipedia.org/wiki/Android_software_development. [Dostopano 9. 8. 2015].

[20] Wikipedia, Gradle - orodje za avtomatsko gradnjo projekta. Dosegljivo: https://en.wikipedia.org/wiki/Gradle. [Dostopano 9. 8. 2015].

[21] Wikipedia, Programski jezik Java. Dosegljivo: https://en.wikipedia.org/wiki/Java_programming_language. [Dostopano 9. 8. 2015].

[22] Wikipedia, Avdio format Waveform - WAV. Dosegljivo: https://en.wikipedia.org/wiki/WAV. [Dostopano 9. 8. 2015].

[23] Wikipedia, JSON. Dosegljivo: https://en.wikipedia.org/wiki/JSON. [Dostopano 9. 8. 2015].

[24] Android Studio. Dosegljivo: http://developer.android.com/tools/studio/index.html. [Dostopano 9. 8. 2015].

[25] AudioTrack dokumentacija. Dosegljivo: http://developer.android.com/reference/android/media/AudioTrack.html. [Dostopano 9. 8. 2015].

Slike

Tabele

Koda